SRA:
Description of the SRA System as Used for MUC–6

George R. Krupka

Systems Research and Applications
4300 Fair Lakes Court
South Building, Suite 500
Fairfax, VA 22033–4232
krupka@sra.com

INTRODUCTION

SRA used the combination of two systems for the MUC–6 tasks: **NameTag**™, a commercial software product that recognizes proper names and other key phrases in text; and **HASTEN**, an experimental text extraction system that has been under development for only one year. For the Named Entity task, SRA adapted a subset of **NameTag**'s capabilities to the MUC–6 specification. For the Template Element task, SRA fed the full results of **NameTag** into **HASTEN**, which performed additional processing to extract and generate the organization and person templates. For the Scenario Template task, SRA fed **NameTag**'s results into **HASTEN**, which used its full extraction capabilities to extract and generate the management succession templates. Figure 1 illustrates the contribution of each system to the MUC–6 tasks. Due to the relative complexity of the scenario template task, this paper will focus on **HASTEN** and the experimental results for scenario extraction, and will provide brief descriptions of **NameTag** and the other tasks.

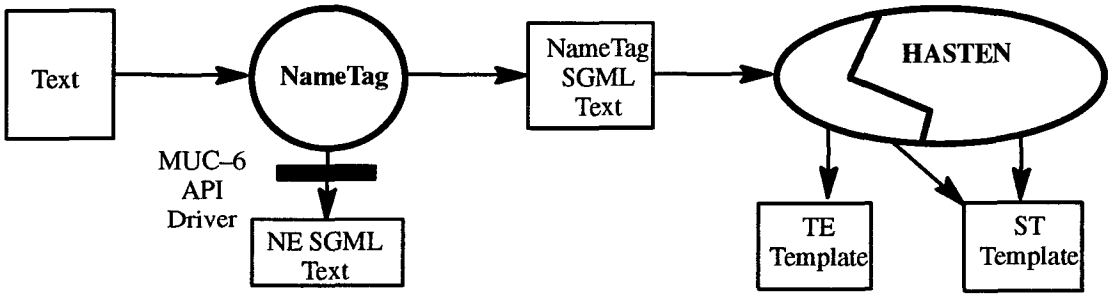

Figure 1: SRA MUC–6 System

SYSTEM DESCRIPTION

Ease of customization, trainability, and automated knowledge acquisition are widely acknowledged as critical issues for text extraction systems. Many systems, including ones from BBN, SRI, Lockheed Martin, University of Massachusetts, and New Mexico State University, have made significant contributions in these areas. SRA has also investigated these issues, and has renewed its effort with the development of **HASTEN**. The long-term objective of **HASTEN** is to provide a system that non-developers can easily customize to extract information from text. The target operating environment might consist of 50 users trying to use **HASTEN** on 50 different extraction scenarios. Therefore, **HASTEN** must be simple, flexible, robust, and trainable, and must minimize the customization effort. These requirements motivate **HASTEN**'s vision of extraction, which is to use the simplest input from users: *extraction examples*. As illustrated in Figure 2, the user annotates examples of what to extract, labelling the important regions of text with their relationship (e.g. the *successee*) to the expressed concept (e.g. *management succession*).

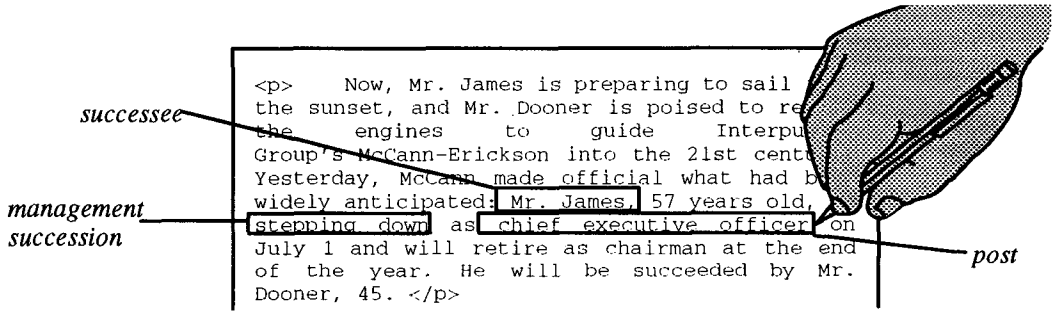

Figure 2: HASTEN's Extraction Vision

HASTEN uses that example to analyze subsequent text. **HASTEN** computes the similarity between an annotated example and the subsequent text, and uses that computation to decide how to analyze it. As more examples are encoded, **HASTEN**'s coverage and accuracy improve.

The **HASTEN** system has a simple architecture, consisting of four main modules shown in Figure 3. The **Analyzer** extracts semantic information from the text, using a set of extraction examples and a supporting knowledge base. The **Reference Resolver** supports the **Analyzer** by providing links from references to their referents. Concept specifications define what concepts to extract, in what order, what semantic roles to fill, and determine how the extraction examples are encoded. The **Collector** collects and merges the semantic information according to concept specifications. The **Generator** uses an output script to convert the collected semantic information into an arbitrary output format, such as a database template. In addition to these core modules, **HASTEN** includes a tokenizer, a document structure facility, a lexical data facility, and an object–oriented (template) scoring program.

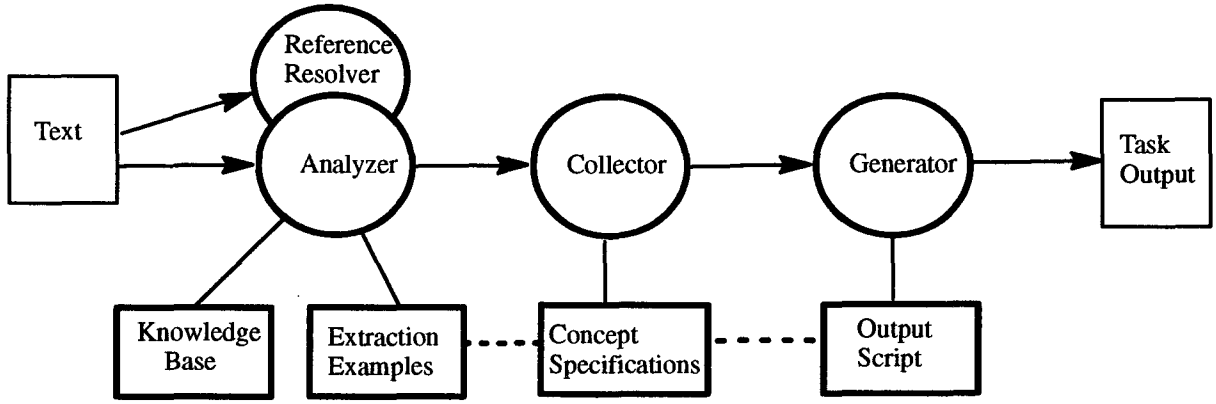

Figure 3: HASTEN System Architecture

Each **Collector** concept represents a processing phase for the **Analyzer**. For each **Collector** concept, the **Analyzer** processes the text and creates a link from the text to the semantic representations, similar to the reference links created by the **Reference Resolver**. Each analysis phase can access the results of previous phases, thus enabling complex embedded semantic representations to be created. For the MUC–6 Scenario Template task, the **Analyzer** first extracted person concepts, then organization concepts, then management post concepts, then succession events.

Extraction by Example

The key module of **HASTEN** is the **Analyzer**, which matches the extraction examples to incoming text and decides what to extract. The **Analyzer** has two components, as shown in Figure 4. The **Matcher** compares an incoming text unit, such as a sentence, to each extraction example. Using a set of parameters, the **Matcher** computes the similarity between them. The **Matcher** also produces an annotated sentence by transferring the extraction annotation from the example to the incoming sentence.

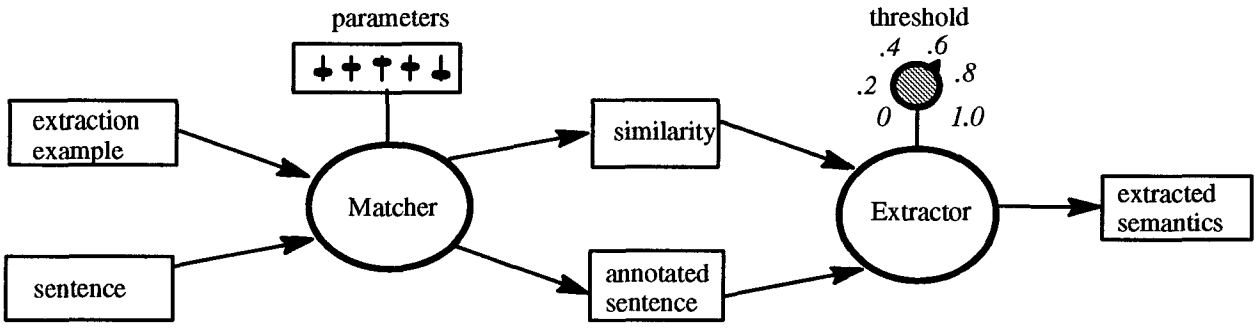

Figure 4: Extraction by Example

The **Extractor** compares the similarity values of all extraction examples, selects the most similar example that exceeds the threshold, and then converts the maximal annotated sentence into a semantic representation. If all extraction examples fail to exceed the threshold, the **Extractor** does not extract anything from the incoming text unit. Thus, a high threshold will result in less extracted semantics than a low threshold. This feature provides **HASTEN** with a central control on extraction performance, which will be illustrated in the test results.

Egraphs

HASTEN represents the extraction examples using a data structure, called an *Egraph*. The major characteristics of this representation are that it is straight–forward to create, conducive to automated learning, easy to compare to each other, and easy to match against text. An Egraph contains three components: an extracted concept, structural elements attached to the original regions of text, and semantic labels attached to the structural elements, as illustrated in Figure 5. In order to use the Egraph for matching other text, the structural element must be generalized into word classes, grammatical constituents, or arbitrarily–defined word sequences. This generalization is represented as a structural class and a set of constraints. The actual definition of the structural class is maintained in the auxiliary knowledge base. For MUC-6, the process of generalizing the structural elements was done manually, using a graphical editor.

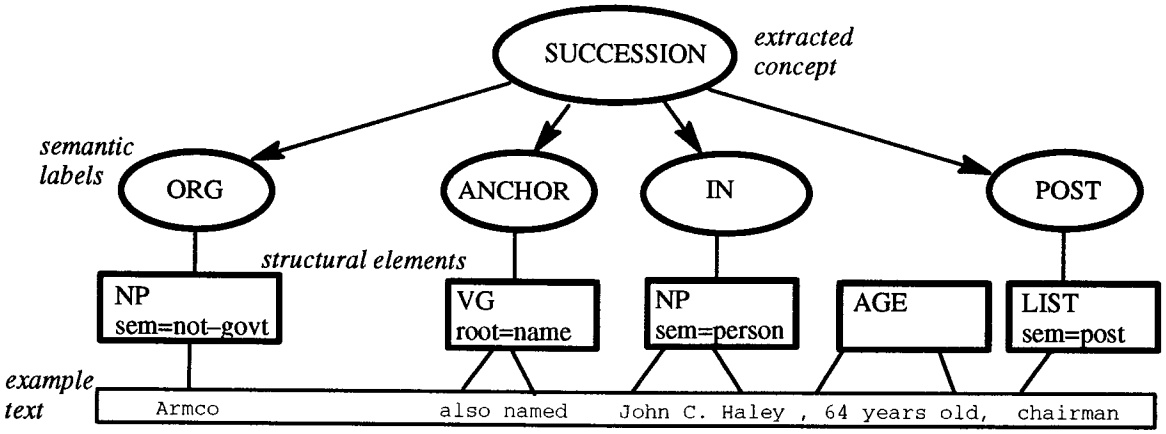

Figure 5: Egraph Structure

In Figure 5, the extracted concept is called SUCCESSION, representing a management succession event. There are five structural elements: a noun phrase with a semantic constraint of a non–governmental organization; a verb group with a head having the root "name"; a noun phrase with a semantic constraint of person; an age phrase; and a list phrase (i.e. a coordinating conjunctive noun phrase) with the semantic constraint of a management post. The semantic labels ORG, IN, and POST are attached to the appropriate structural elements. The semantic label ANCHOR is attached to the main element, or head, of the example.

Similarity Metric

HASTEN uses a metric to compute the similarity between an Egraph and an incoming text unit. **HASTEN** first matches the structural elements, and binds the semantic labels of those elements that successfully matched. The metric computes the percent of the Egraph that matched, using a weighted sum of factors. The factors represent how well the elements match, how well they are ordered, how well the adjacent elements are joined, and how much semantic content was bound. The weights are configuration parameters that can be adjusted. A perfect match results in a 1.0 similarity value. An incoming text with absolutely nothing in common with the Egraph receives a 0.0 similarity value.

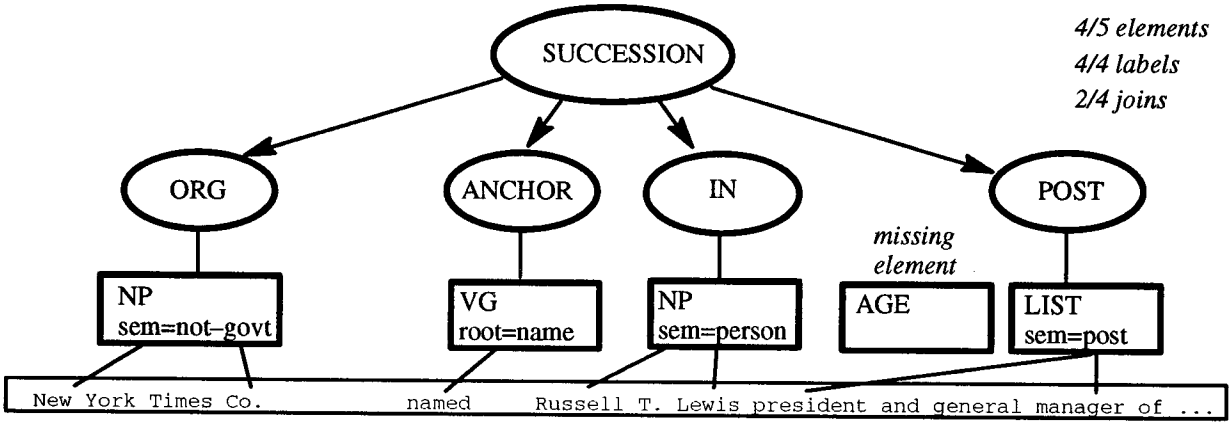

Figure 6: Egraph Matching

Figure 6 illustrates the comparison of an Egraph to a textual unit. The incoming text does not match the AGE structural element, causing a failure to join the person NP and the post LIST elements. However, the four semantic

labels are successfully bound. Therefore, this incoming text matches 4 of the 5 structural elements, 4 of the 4 semantic labels, and joins 2 of the 4 adjacent element pairs. This Egraph will receive a high similarity value, with its exact value depending on the weights.

Training and Parameterization

During development, **HASTEN** may be run on training texts that have been used to create Egraphs. However, the Egraphs created from a given text unit are withheld and not used for extraction on that text unit. This practice ensures that **HASTEN** has no knowledge about a particular incoming text, and that the training set is not corrupted. However, these withheld Egraphs can be treated as a *key* for evaluating the other Egraphs. **HASTEN** has a training module that runs each Egraph against every text unit in the training text, and for those units that have an Egraph key, recall and precision measures are computed based on the semantic labels. Egraphs that match a text unit that has no Egraph key are spurious. The training module collects the results and produces tables of results. The first table is a listing of every text unit, the maximal matching Egraph, the similarity value, whether the similarity exceeded the threshold (i.e. the `Result`), and the performance (i.e. F–measure for semantic labels). The Egraphs are named after the originating document. The `Config` column lists the name of the particular settings of similarity metric weights and threshold used during the matching, in this case `essential-70`. Table 7 shows a portion of this listing.

File	Config	Component	Egraph	Match	Result	Performance
9404250043	essential-70	SENTENCE[2]	930219-0013	0.86	Succeeded	0.00
9404250043	essential-70	SENTENCE[2]	930219-0013	0.86	Succeeded	0.00
9404250043	essential-70	SENTENCE[2]	930219-0013	0.86	Succeeded	100.00
9404250043	essential-70	SENTENCE[4]	931028-0075.E	0.78	Succeeded	75.00
9404250043	essential-70	SENTENCE[5]	930219-0013.B	1.00	Succeeded	100.00
9404220086	essential-70	SENTENCE[5]	940131-0163	0.65	Failed	66.67
9404220086	essential-70	SENTENCE[6]	930219-0013.B	0.99	Succeeded	0.00
9404220086	essential-70	SENTENCE[6]	930219-0013.B	0.99	Succeeded	66.67
9404190012	essential-70	SENTENCE[5]	930225-0042	0.88	Succeeded	13.33
9404190012	essential-70	SENTENCE[6]	940412-0147	0.85	Succeeded	50.00
9404180007	essential-70	SENTENCE[5]	930219-0013.B	0.99	Succeeded	30.77

Table 7: Egraph Training Results

The second table is a list summarizing the training information for each Egraph. This listing contains the name of the configuration, the number of matches, the number of spurious matches, the average similarity value for all matches, the total extraction recall, and the over–generation (i.e. spurious over count). Table 8 shows a portion of this listing.

Config	Egraph	Count	Spurious	Match	Recall	Overgeneration
essential-70	930219-0013.B	31	7	0.96	30.15	0.23
essential-70	940425-0043.A	31	26	0.81	23.81	0.84
essential-70	930219-0013.A	24	20	0.92	18.00	0.83
essential-70	930219-0013.E	21	2	0.98	40.87	0.10
essential-70	930628-0011.B	16	8	0.95	32.50	0.50
essential-70	930629-0071.B	15	12	0.79	45.00	0.80
essential-70	931028-0075.B	13	7	0.79	15.00	0.54
essential-70	930412-0090	12	9	0.86	16.67	0.75
essential-70	940412-0147.A	10	7	0.80	44.74	0.70
essential-70	930219-0013	9	2	0.86	35.71	0.22

Table 8: Egraph Training Summary

Since multiple parameter configurations can be run, the **HASTEN** training module provides valuable insight into what weights will achieve the best result and what threshold value will maximize extraction performance.

Extraction Bias

Another benefit of the training module is that it provides an additional parameter that can be factored into the similarity metric. After the training module has run, the results can be probed to construct an *extraction bias*, which consists of a number from 0 to 1.0 for each Egraph. The similarity value is multiplied by this number to adjust the result. A bias of 0 effectively disables the Egraph, a bias of 1.0 leaves the original value unchanged, and a bias in between reduces the value. The simplest bias is one that disables some of the Egraphs, and leaves the other Egraphs unchanged. This bias is useful since it can eliminate the Egraphs that predominately over–generate. A complex bias might adjust each Egraph similarity value to favor high–frequency or high–precision Egraphs. Several test runs using extraction biases are presented in the Test Results section.

Extraction Links

The **Extractor** creates semantic representations based on the Egraph matches. However, the Egraph match is simply an annotated text unit, with semantic labels bound to portions of the incoming text. For example, the `ORG` label

might be bound to "the company" or "IBM", and the **Collector** is expecting an embedded semantic representation to fill the ORG role. To accomplish this, the **Extractor** has access to *links* that either connect the references to their referents or connect named entities to their semantic representation. The **Reference Resolver** creates the links for references, and the **Extractor** itself creates links from earlier processing passes. A portion of text that has no links is considered a generic reference, and a special unnamed semantic representation is created dynamically.

CUSTOMIZATION

HASTEN is simple to customize, and involves the steps listed below. Due to the complexity of the task, these steps may be repeated to adjust the definitions.

- Encode the template specification according to the task specification. This step not only supports the creation of the **Generator** output scripts, but also enables the loading of answer key templates for analysis and evaluation.
- Define the **Collector** concept specifications, including semantic roles and constraints.
- Define the **Generator** output script which maps the **Collector** concepts to the template format. The output script invokes utility functions to convert the **Collector** data structure into the template format. Depending on task specification, special purpose routines may be required.
- Formulate Egraphs for examples from the training texts. Encoding an Egraph requires less than a minute, using a graphical editor. However, for MUC-6, significant time was spent comprehending the templates and locating the originating text units.

The remainder of the effort involves determining the similarity metric weights and thresholds to maximize the extraction performance.

The Template Element task required 2 PERSON Egraphs, one for an untitled personal name and one for a titled personal name. The Template Element task required 44 ORGANIZATION Egraphs, in order to extract the locations, nationalities, local descriptors, and unnamed organizations. The Scenario Template task required 132 SUCCESSION Egraphs. In total, the Egraphs referenced 12 structural element classes (e.g. NP), and were constrained to form 100 unique structural elements. The Egraphs required 14 syntactic categories, 20 semantic classes, and 2 lexical properties.

SYSTEM WALKTHROUGH

This section will provide a brief description of **HASTEN**'s performance on the selected walkthrough document. **HASTEN** performed reasonably well, achieving a recall/precision of 38/77. **HASTEN** extracted the principal succession event involving "James" and "Dooner," but failed to detect both management posts. **HASTEN** failed to extract a secondary succession event involving "Kim."

Analysis

For each text unit, the **Analyzer** compared the SUCCESSION Egraphs, computed the similarity metric value, and selected the maximal matching Egraph that exceeded the similarity threshold. The first successful match occurred in the headline, resulting in the extraction of the succession event, but not the post:

```
INPUT: "Marketing & Media--Advertising: John Dooner Will Succeed James At
Helm of McCann- Erickson"
EXAMPLE: 930219-0013.B  (similarity 1.0) "He succeeds Lance R. Primis"
COLLECT: #<SEM :SUCCESSION 747>
             :IN #<SEM :PERSON 2441 :NAME "John Dooner" >
             :OUT #<SEM :PERSON 2442 :NAME "James">
```

The **Analyzer** had created the semantic PERSON representations during a previous processing phase, and linked them to the originating text. The **Analyzer** accesses these representations and fills the :IN and :OUT slots. The next match occurred in sentence 2, resulting in the additional extraction of the organization and post:

```
INPUT: "Yesterday, McCann made official what had been widely anticipated:
Mr. James, 57 years old, is stepping down as chief executive officer on
July 1 and will retire as chairman at the end of the year. "
EXAMPLE: 940128-0022 (similarity .86) "E-Systems Inc. said E. Gene Keiffer
stepped down as chief executive officer"
COLLECT: #<SEM :SUCCESSION 744>
             :ORG  #<SEM :ORGANIZATION 2351 :NAME "McCann">
             :OUT #<SEM :PERSON 2391 :NAME "James">
             :POST "chief executive officer "
```

Note that the semantic PERSON representation for "James" has a different identifier (i.e. 2391) than the representation from the headline (i.e. 2442). It is the **Collector**'s responsibility to merge identical or compatible representations. The **Collector** will also merge the SUCCESSION representations from the headline and sentence 2, as described in the next section. The next match occurred in sentence 3, resulting in the erroneous extraction of the succession event in reverse. The example was encoded without regard to the active/passive feature, and therefore, the only structural difference between the example and the input is that the input has the preposition "by," thus resulting in the near–perfect similarity value of 0.99. If the **Analyzer** had a passive example, its Egraph would have matched perfectly and therefore pre–empted this erroneous match. In this sentence, the **Reference Resolver** correctly resolved the pronoun "He" to the last mention of "James", thus resulting in the extraction of the PERSON 2391 representation for "James."

```
INPUT: "He will be succeeded by Mr. Dooner, 45."
EXAMPLE:930412-0090.A(similarity.99) "He succeeds investor Carl H.Lindner"
COLLECT: #<SEM :SUCCESSION 745>
           :OUT #<SEM :PERSON 2393 :NAME "Dooner">
           :IN #<SEM :PERSON 2391 :NAME "James"  :REF "He">
```

Sentence 20 contained the succession event for "Kim." The training examples did not contain a sentence involving the word "hire," and thus the Egraphs were not similar enough to result in a match. The closest examples achieved a similarity value of approximately 0.60.

```
In addition, Peter Kim was hired from WPP Group's J. Walter Thompson last
September as vice chairman, chief strategy officer, world-wide.
```

Collection

The **Collector** receives the semantic representations from all the sentences, and merges them into a cumulative semantic representation. The **Collector** maintains separate semantic representations for incompatible information. In the walkthrough document, the **Collector** combines the semantic representations from the headline and sentence 2 into the following representation:

```
#<EXT :SUCCESSION 56>
   :IN #<EXT  :PERSON 373 :NAME "John Dooner"/"Dooner"/"JohnJ.Dooner Jr."
                          :TITLE "Mr.">
   :ORGANIZATION #<EXT :ORGANIZATION 211 :NAME "McCann-Erickson"/"McCann">
   :OUT #<EXT :PERSON 374 :NAME "James"/"Robert L. James" :TITLE "Mr.">
   :POST "chief executive officer ">
```

Multiple references to the same named entity (e.g. "James" and "Robert L. James") are merged, relying on the alias information provided by **NameTag**. The PERSON Egraphs extract and fill the :TITLE slot. The erroneous SUCCESSION representation from sentence 3 is incompatible with this structure, and is maintained separately.

Generation

The **Generator** applies an output script to the **Collector** representations to produce the data templates. Since the erroneous succession event from sentence 3 does not have a :POST fill, the output script invalidates it and no template is generated. For object–oriented templates used in MUC–6, the output script must recursively traverse the **Collector** representations and apply conversion routines for each sub–template. The **Generator** actually produces a *template* data structure, which can be easily printed, but also fed directly to **HASTEN**'s scoring program. The scoring program employs a top–down comparison algorithm that produces performance measures as well as a side–by–side display, as illustrated below. The display shows the individual credit assignments as well as the recall/precision subtotals for each object:

```
          RESPONSE                                        KEY
-------------------------------------------------------------------------------
1/1 <TEMPLATE-9402240133-1>                  1/3 <TEMPLATE-9402240133-1>
     CONTENT                                      CONTENT
       *** MISSING FILL ***                        0 <SUCCESSION_EVENT-9402240133-3>
       *** MISSING FILL ***                        0 <SUCCESSION_EVENT-9402240133-2>
     <SUCCESSION_EVENT-9402240133-1>             1 <SUCCESSION_EVENT-9402240133-1>
4/5 <SUCCESSION_EVENT-9402240133-1>          4/5 <SUCCESSION_EVENT-9402240133-1>
     VACANCY_REASON                               VACANCY_REASON
       REASSIGNMENT                                 0 DEPART_WORKFORCE
     IN_AND_OUT                                   IN_AND_OUT
       <IN_AND_OUT-9402240133-1>                    1 <IN_AND_OUT-9402240133-2>
       <IN_AND_OUT-9402240133-2>                    1 <IN_AND_OUT-9402240133-1>
     POST                                         POST
       "\"chief executive officer\""                1 "\"chief executive officer\""
     SUCCESSION_ORG                               SUCCESSION_ORG
       <ORGANIZATION-9402240133-1>                  1 <ORGANIZATION-9402240133-1>
2/5 <IN_AND_OUT-9402240133-2>                2/3 <IN_AND_OUT-9402240133-1>
     ON_THE_JOB                                   ON_THE_JOB
       NO                                           0 YES
     NEW_STATUS                                   NEW_STATUS
       OUT                                          1 OUT
     IO_PERSON                                    IO_PERSON
       <PERSON-9402240133-2>                        1 <PERSON-9402240133-2>
     OTHER_ORG                                    *** SPURIOUS SLOT ***
       <ORGANIZATION-9402240133-1>
     REL_OTHER_ORG                                *** SPURIOUS SLOT ***
       SAME_ORG
4/5 <IN_AND_OUT-9402240133-1>                4/5 <IN_AND_OUT-9402240133-2>
     REL_OTHER_ORG                                REL_OTHER_ORG
       SAME_ORG                                     1 SAME_ORG
     OTHER_ORG                                    OTHER_ORG
       <ORGANIZATION-9402240133-1>                  1 <ORGANIZATION-9402240133-1>
     ON_THE_JOB                                   ON_THE_JOB
       UNCLEAR                                      0 NO
     NEW_STATUS                                   NEW_STATUS
       IN                                           1 IN
     IO_PERSON                                    IO_PERSON
       <PERSON-9402240133-1>                        1 <PERSON-9402240133-1>
2/3 <ORGANIZATION-9402240133-1>               2/4 <ORGANIZATION-9402240133-1>
     ORG_TYPE                                     ORG_TYPE
       OTHER                                        0 COMPANY
     ORG_ALIAS                                    ORG_ALIAS
       "\"McCann\""                                 1 "\"McCann\""
     ORG_NAME                                     ORG_NAME
       "\"McCann-Erickson\""                        1 "\"McCann-Erickson\""
     *** MISSING SLOT ***                         ORG_DESCRIPTOR
                                                    0 "\"one of the largest...
4/4 <PERSON-9402240133-1>                    4/4 <PERSON-9402240133-1>
     PER_TITLE                                    PER_TITLE
       "\"Mr.\""                                    1 "\"Mr.\""
     PER_ALIAS                                    PER_ALIAS
       "\"Dooner\""                                 1 "\"Dooner\""
       "\"John Dooner\""                            1 "\"John Dooner\""
     PER_NAME                                     PER_NAME
       "\"John J. Dooner Jr.\""                     1 "\"John J. Dooner Jr.\""
3/3 <PERSON-9402240133-2>                    3/3 <PERSON-9402240133-2>
     PER_TITLE                                    PER_TITLE
       "\"Mr.\""                                    1 "\"Mr.\""
     PER_ALIAS                                    PER_ALIAS
       "\"James\""                                  1 "\"James\""
     PER_NAME                                     PER_NAME
       "\"Robert L. James\""                        1 "\"Robert L. James\""
```

TEST RESULTS AND ANALYSIS

For the formal MUC-6 test data, **HASTEN** had three official configurations: one to maximize recall, one to maximize precision, and one to maximize both. A simple adjustment to the similarity metric threshold created these configurations. The training module determined the values of the thresholds, and also determined the optimal extraction bias, which disabled the most over-generating Egraphs. Figure 9 shows the results of the three official configurations for both the training and the test data, as well as additional data points for other threshold settings. This figure clearly illustrates that **HASTEN** has the ability to trade recall for precision.

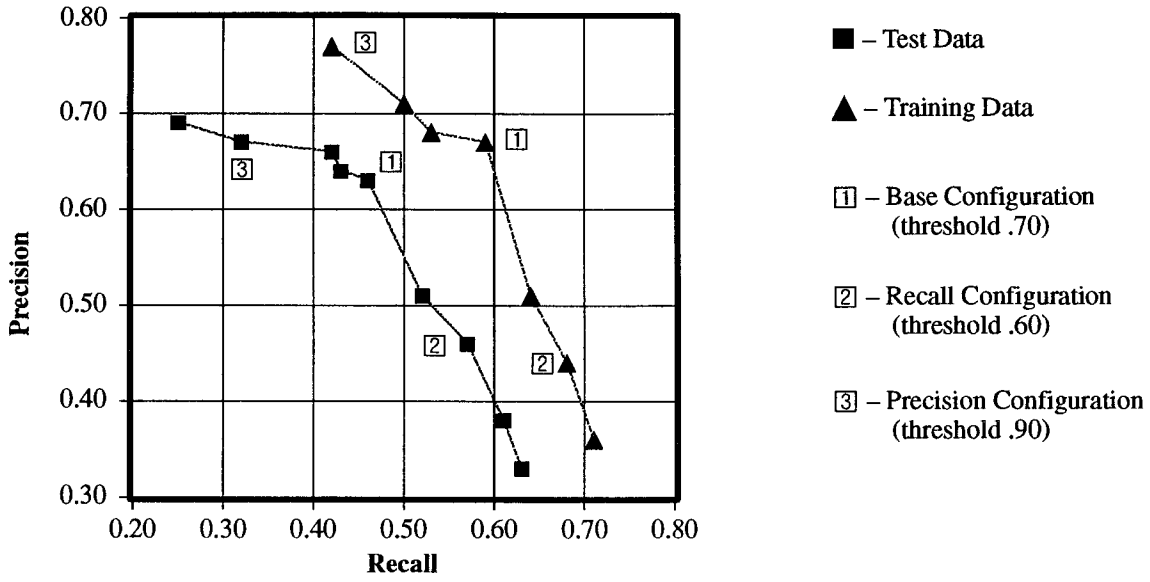

Figure 9: Scenario Template Test Results

HASTEN rapidly achieved its extraction performance, as illustrated in Figure 10. After the initial effort to encode the training examples, the training module determined the optimal similarity metric parameters (see [2]). During this time, no effort was made to actually generate the template slots `VACANCY_REASON` and `ON_THE_JOB`, and **HASTEN** generated a default fill of `UNCLEAR` and `NO`, respectively. **HASTEN** also defaulted the `OTHER_ORG` to be the same as the `SUCCESSION_ORG`, and therefore `REL_OTHER_ORG` was always `SAME_ORG`. A few days were then spent on those slots, raising the performance slightly (see [3]). **HASTEN**'s performance got a boost from the latest upgrade to the scoring program and keys (see [4]). The remainder of the test period was spent on improving the name recognition, which impacts all three tasks, but resulted in very little improvement on the scenario template task.

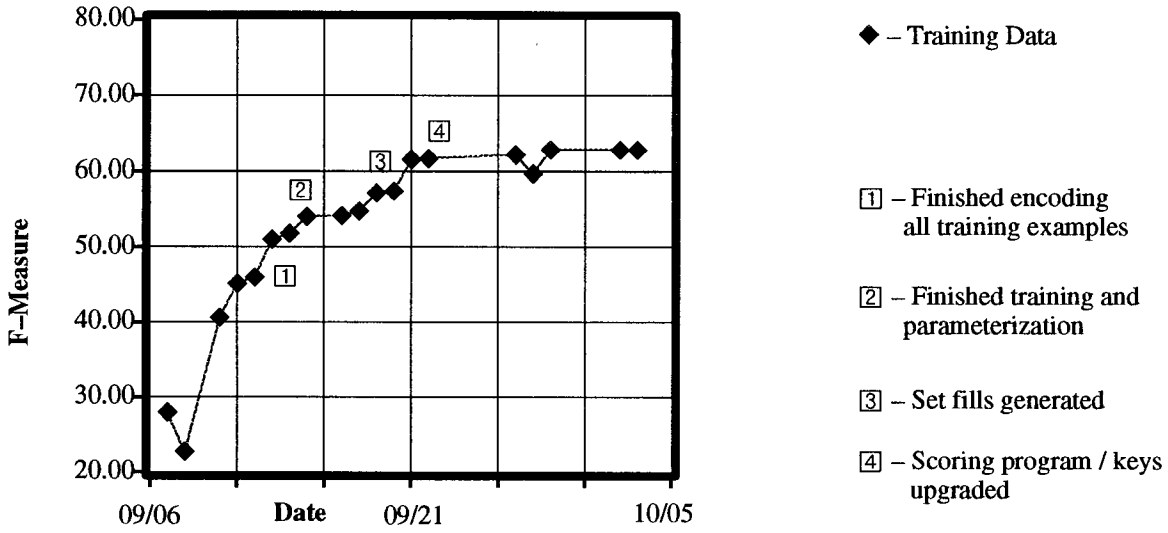

Figure 10: Extraction Performance Time Line

EXPERIMENTAL RESULTS

Even though the MUC–6 extraction task focused on one scenario, SRA did not want to produce a single extraction result. SRA's focus was on experimentation with MUC–6 providing a testing environment. This section reports on various other test results that fall outside of the official MUC–6 tests.

Example Sampling

Since extraction examples are the core knowledge source for **HASTEN**'s extraction capability, it is worthwhile to explore the relationship between the number of examples and extraction performance. Furthermore, the order of encoding the examples may also effect performance.

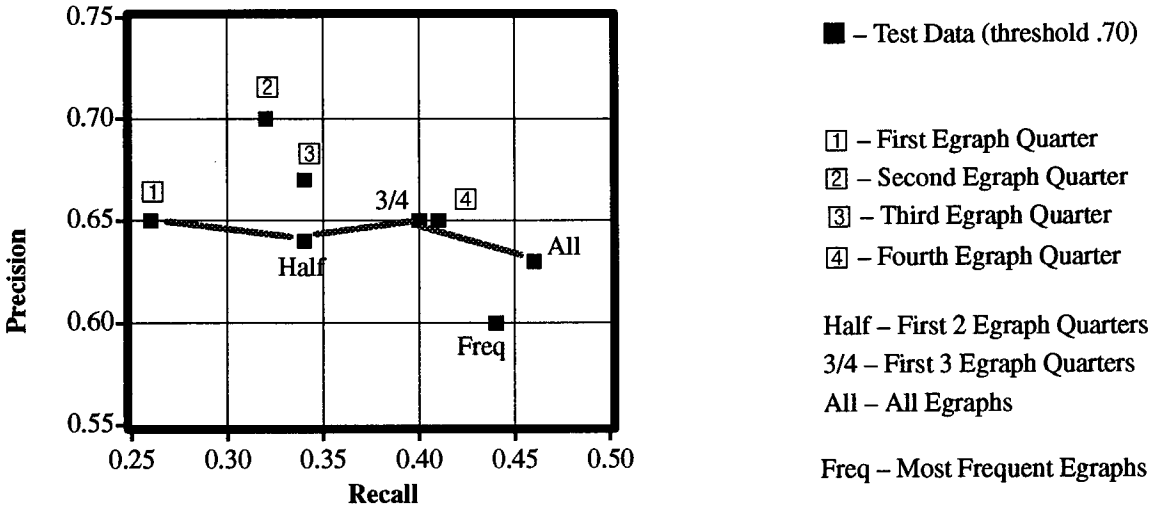

Figure 11: Egraph Subset Test Results

Figure 11 shows eight experimental runs on the final test data, using extraction biases to partition the Egraphs in several ways. The first four runs use one quarter (33) of the total number of `SUCCESSION` Egraphs (132), which were sequentially ordered by the document number of the originating text unit. The next three runs use one–half, three–quarters, and all of the Egraphs, respectively. The first quarter run is connected with these three runs to show the gradual improvement in recall (26 to 46) and the minor degradation in precision (65 to 63) as more examples were given to **HASTEN**. Notice that the fourth Egraph quarter out–performs the first three quarters. Presumably, the fourth Egraph quarter includes generally applicable examples, while the first three Egraph quarters include unusual or redundant examples. The last run, labelled `Freq`, consists of running only those Egraphs that matched at least two training text units (42 total), approximately one third of the total Egraphs. Presumably, this configuration eliminates the unusual and redundant examples, and produces the performance near the level of all Egraphs.

Alternative Metric Weights

The Egraph similarity metric utilizes a weighted sum of factors. The official MUC–6 test results considered only one configuration of weights, which created a strong preference for the semantic content, especially the `ANCHOR` label. As an alternative, **HASTEN** was configured with weights that created a strong preference for the structural match. This experiment did not produce significantly different results than the official configuration, as illustrated in Figure 12. The five point drop in recall for the BASE configuration does demonstrate that structural differences in examples may interfere with the extraction of semantic content.

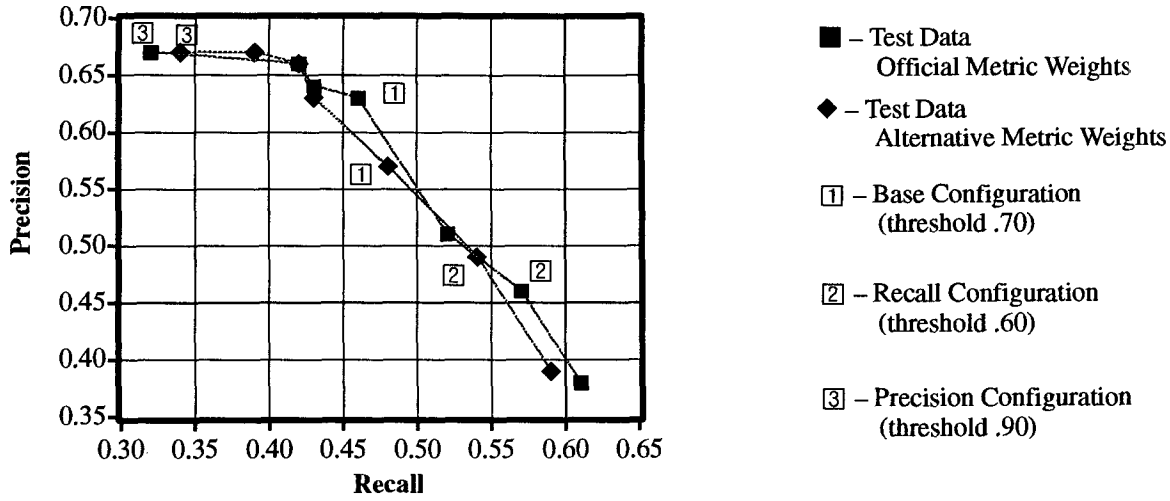

Figure 12: Alternative Metric Weights Experiment

Extraction by Hand

As described earlier, the Egraphs originating from a particular text unit are withheld and not used for extraction on that unit. However, **HASTEN** has a special mode that runs only the Egraph keys, in order to test the rest of the system. This mode can test how well the **Collector** is merging information, how well the **Reference Resolver** is working, or test the format produced by the **Generator**. In effect, this mode simulates perfect Egraph matching, and predicts the upper bound on extraction performance for those Egraphs.

The Egraph key extraction performance is not 100% recall and precision, for a variety of reasons. First, not all occurrences of the extraction concept can be encoded with an Egraph. Elliptical or other highly contextual references can not be feasibly encoded. Second, mistakes in reference resolution can cause the extraction of erroneous semantic representations, even if the Egraph match is correct. Third, the **Collector** can erroneously merge or split the extracted semantic representations, even if the Egraph matches are independently correct. Fourth, due to the task specification, some of the scenario extraction output may not come directly from the Egraph matches. Fifth, for MUC–6, about 25% of the management scenario template fills are contained in the `PERSON` and `ORGANIZATION` objects.

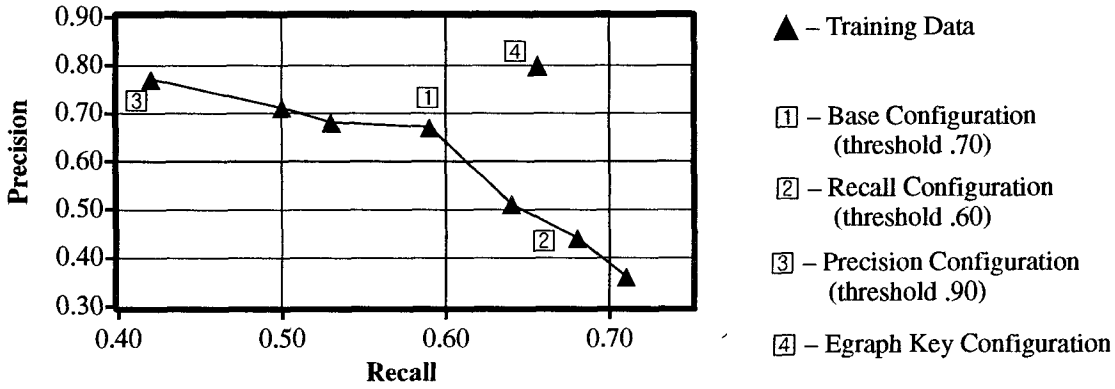

Figure 13: Egraph Key Performance

Figure 13 shows the results of running the Egraph keys for the training data, in relation to the other configurations. The significant improvement to precision is due to the elimination of all spurious Egraph matches, since Egraph keys are by definition relevant. Recall also improves mainly because unusual examples that fail to match other Egraphs are now matched by their own Egraph.

Micro–MUC

The MUC–6 Scenario Template framework achieved its goal of reducing the amount of task–specific and domain–specific customization required by the extraction systems. However, the succession event scenario still

included a few template slots that forced systems to attempt to make subtle and inferential judgements; namely, the VACANCY_REASON, ON_THE_JOB, and REL_OTHER_ORG slots. Furthermore, template specification itself was rather cumbersome due to the IN_AND_OUT object, which really was a "pseudo" object for grouping related information. These features of the task specification confused the customization and evaluation of extraction systems on the central scenario event, namely the management succession. Therefore, as an additional experiment, SRA devised the very minimal (or *micro–MUC*) template specification to represent the management succession event, as shown below:

```
<SUCCESSION_EVENT> :=
     POST: "post"
     IN:  <PERSON>+
     OUT: <PERSON>+
     ORG: <ORGANIZATION>
```

This template specification completely eliminates the IN_AND_OUT object, the set fill slots, and the distinction of an acting post. This specification required changes to only **HASTEN**'s **Generator** script. To evaluate its performance, SRA automatically converted the answer keys, and edited the official scoring program configuration file.

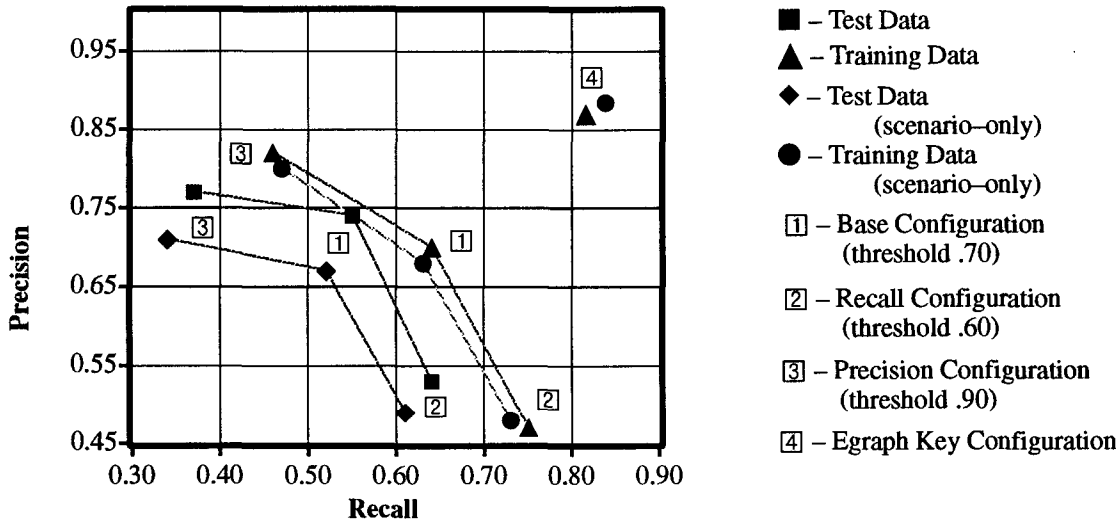

Figure 14: micro–MUC Scenario Template Test Results

Figure 14 shows the performance results for the training and test data. This figure also shows the adjusted performance results (labelled *scenario–only*) that factor out the slots of the PERSON and ORGANIZATION objects, since these slots confuse the evaluation of the scenario event extraction. The points labelled [4] are the results of running the Egraph keys on the training data. These points represent the potential extraction performance on the central scenario event.

Labor Negotiation Scenario Template Results

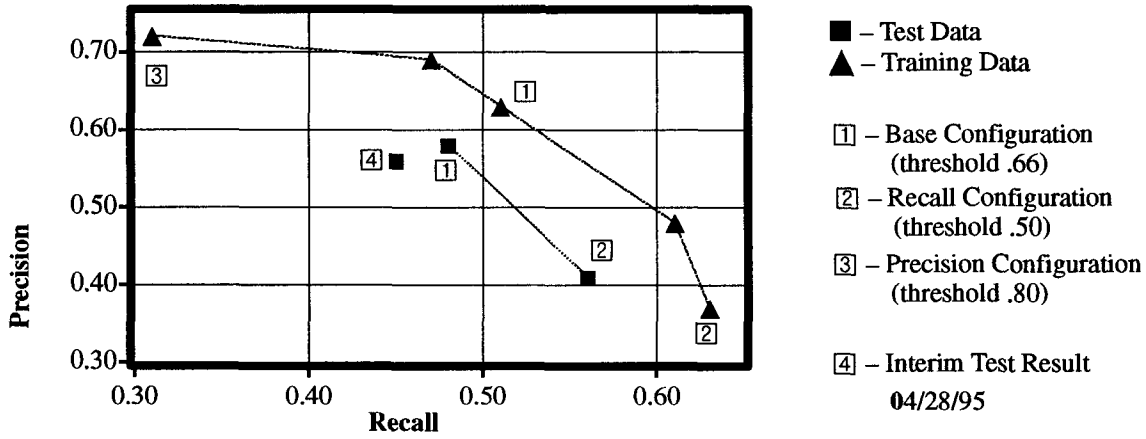

Figure 15: Scenario Template Interim Results

The generality of **HASTEN**'s design can only be tested by using other task definitions in other domains. The MUC–6 interim scenario task of labor negotiations provides another good application for **HASTEN**. Figure 15 shows the final performance results on the labor negotiation data.

Egraph Mutations

Since the ultimate goal of **HASTEN** is to minimize the customization effort, **HASTEN** must strive to maximize its performance from as few examples as possible. One possibility is to automatically derive other Egraphs from those that have been encoded manually. **HASTEN** has another special module that tries to *mutate* Egraphs in a variety of ways, based on their similarity with other Egraphs. Currently, there are three mutation methods:

- *cross–over* – replace some structural elements of one Egraph with elements from another;
- *trim* – eliminate structural elements from the ends of the Egraph;
- *merge* – combine the structural elements of multiple Egraphs;

The mutation module compares every Egraph with each other, and for each pair of significantly similar Egraphs, applies the three methods. The resulting Egraphs are saved and can be treated in the same way as a manually created Egraph. Thus, the training module can run these derivative Egraphs to determine how well they perform, and construct an extraction bias to include best ones. This module is promising, but there was insufficient time to fully investigate it for the MUC–6 evaluation.

HASTEN IMPLEMENTATION

HASTEN is implemented in Allegro Common LISP, including a development environment written in CLIM. **HASTEN** consists of 12,675 lines of code, and the development environment consists of 12,450 lines of code. **HASTEN** required approximately 20 person–weeks for its development, and MUC–6 required 16 person–weeks of effort, including the interim test, formal test, and final report. Table 16 shows the processing time for the three official Scenario Template configurations, run on a Sun SPARCstation 20. The 100 final test data documents have an original size of 261,658 characters. **NameTag** took 19 seconds to process these documents, as shown in the **CPU Time** column.

Configuration	CPU Time (seconds)	Speed (Meg/hour)	Real Time (minutes)	F–Measure
BASE	334 (+ 19)	2.67	12	53.27
RECALL	525 (+ 19)	1.73	18.6	50.98
PRECISION	310 (+ 19)	2.86	11.2	43.24

Table 16: Scenario Template Processing Statistics

NAMED ENTITY TASK

SRA performed the Named Entity task using its commercial name recognition product, called **NameTag**™. **NameTag** is a high–speed software program consisting of a C++ engine and name recognition data. **NameTag** uses its own tag specification that classifies names and other key phrases, and can either generate SGML annotated text or a table of extracted entities. Besides the classification, **NameTag** also assigns unique identifiers to those names that refer to the same entity, such as "International Business Machines" and "IBM." **NameTag** also assigns country codes to place names. **NameTag** required 20 person–weeks for its engine development, 9 person–weeks for its data, and 10 person–weeks for its development interface and utilities.

NameTag has three major processing modes that represent trade–offs between performance and speed. The BASE configuration performs the maximum analysis, achieves the best results, but is the slowest. The FAST mode reduces the analysis to increase speed with minimal degradation in performance. The FASTEST mode performs the minimum analysis at the greatest speed with the lowest performance. **NameTag** includes a small number of personal and organization names (currently 530), which eliminate the need to dynamically recognize them. **NameTag** can process text without the use of these names. **NameTag** also can be run in case–insensitive mode to handle text in all upper case.

For the MUC–6 Named Entity task, a 300 line C++ driver program used the **NameTag** API to run its name recognition, access the table of extracted entities, map the **NameTag** classification into the MUC–6 specification, and generate the SGML annotated document. Since **NameTag** recognizes more names and phrases than defined for MUC–6, such as publications and relative temporal expressions, the driver program filtered some extracted entities. Since the links between aliases are not required for the Named Entity task, the driver suppressed this **NameTag** information.

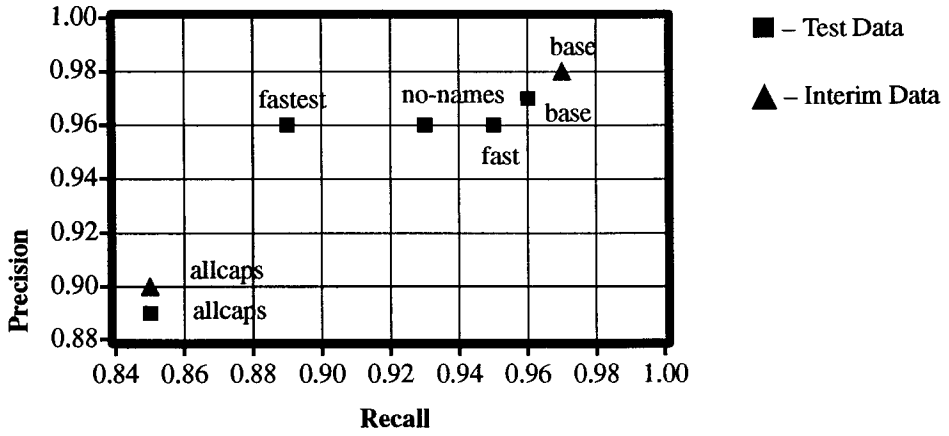

Figure 17: Named Entity Test Results

SRA submitted four official configurations: the BASE configuration, the two speed configurations, and the configuration without the use of personal and organizational names. Figure 17 shows the name recognition performance for the final test data, plus two reference points using the interim test data. SRA also conducted a test run using the case–insensitive mode, which is labelled *allcaps* in the figure. The **NameTag** case–insensitive mode was run on the upper–case version of the test data. Since the test data was manually tagged in mixed case and the MUC–6 task specification includes case–sensitive tagging rules, the case–insensitive performance would actually be slightly higher. For example, in mixed case, "group" is not included in the tag for "Chrysler group." However, in upper case text, "GROUP" would presumably be included in the tag.

Configuration	Rules	CPU Time (seconds)	Speed (Meg/hour)	F–Measure
BASE	226	3.72	78.68	96.42
FAST	86	3.33	87.73	95.66
FASTEST	59	2.62	111.76	92.61
NO-NAMES	226	3.67	79.75	94.92

Table 18: Named Entity Processing Statistics

Table 18 shows the processing time and speed of the four official configurations for the Named Entity test data, run on a Sun SPARCstation 20. The 30 test documents had a size of 87,203 characters. The number of recognition rules for each configuration is also shown. Table 19 contains the performance measures for the ENAMEX tag and its sub–classifications. The ENAMEX tag is the most difficult due to the ambiguity between people, places, and organizations. Furthermore, case sensitivity is more significant in the recognition of these names, as opposed to the numeric and temporal entities. Also, the NO-NAMES configuration excluded the use of personal and organizational names.

Configuration	Organization	Person	Location	Enamex
BASE	91 / 94	98 / 99	99 / 95	96 / 96
FAST	90 / 96	98 / 98	96 / 91	94 / 96
FASTEST	87 / 96	96 / 98	96 / 98	92 / 97
NO-NAMES	85 / 92	97 / 99	99 / 94	92 / 96
ALLCAPS	72 / 83	95 / 95	92 / 90	82 / 87

Table 19: ENAMEX Performance Measures (Recall/Precision)

The three speed configurations show the general trade–off between speed and recall, with precision remaining about the same. Note the anomaly in Location precision measure for the fast configuration; this is a side–effect of recognizing less organizations, which pre–empt the location classification. The NO–NAMES configuration had little effect on person names, illustrating how simple they are to dynamically recognize. The NO–NAMES configuration

resulted in a significant drop in recall for organization names, reflecting the references to *household names* with little contextual clues, such as "Microsoft."

System Walkthrough

NameTag performed very well on the selected walkthrough document, achieving a recall/precision of 98/97 for the BASE configuration, resulting from three errors. **NameTag** classified "J. Walter Thompson" as a person rather than a organization, since it *looks* like a personal name. **NameTag** tagged "Coca–Cola" within "Coca–Cola Classic" as an organization, since it failed to recognize the larger product name. **NameTag** tagged "Goldman" within "Kevin Goldman" as an organization, since that is a company alias in its static list of names. The NO–NAMES configuration eliminates this error, but causes **NameTag** to miss the mentions of "Coke" and "Coca–Cola," which were also contained in its static list of names.

The FAST configuration drops the performance to 87/96 due to its failure to learn "McCann–Erickson" as an organization, which consequently causes **NameTag** to miss 9 "McCann" aliases. The FASTEST configuration further drops performance to 82/94, since it does not apply the ampersand recognition rule to find "Ammirati & Puris" as an organization, and then individually tags "Puris" as an alias to "Martin Puris." The ALLCAPS configuration achieves 92/92, due to additional erroneous names "BIG HOLLYWOOD TALENT AGENCY," "JAMES PLACES," "DOONER DECLINES," and "COKE ADVERTISING."

TEMPLATE ELEMENT TASK

SRA combined the results of **NameTag** and some additional processing by **HASTEN** to perform the Template Element task. **NameTag** performs the majority of the work, since it identifies the person and organization names, classifies them, and resolves the aliases. **NameTag** also assigns country codes to the place names which support the normalized ORG_COUNTRY slot. **HASTEN** matched person and organization Egraphs to extract additional local information, such as location, nationality, and descriptors. The **Reference Resolver** attempted to resolve organizational references to the names in order to extract additional non-local information.

SRA submitted two official configurations: a BASE configuration running all components of **HASTEN**; and a NO-REF configuration with the **Reference Resolver** disabled. Since the **Reference Resolver** provides some of the organizational descriptors, which may include some location or nationality information, the second configuration resulted in lower recall. However, since reference resolution is difficult, erroneous references can hurt precision. To demonstrate the portion of **HASTEN**'s performance comes from **NameTag**, a third unofficial configuration was run that disabled reference resolution and the extraction of locations, nationalities, and descriptors. Figure 20 shows the performance results for these three configurations on the final test and training data, as well as the BASE configuration performance on the interim test and training data.

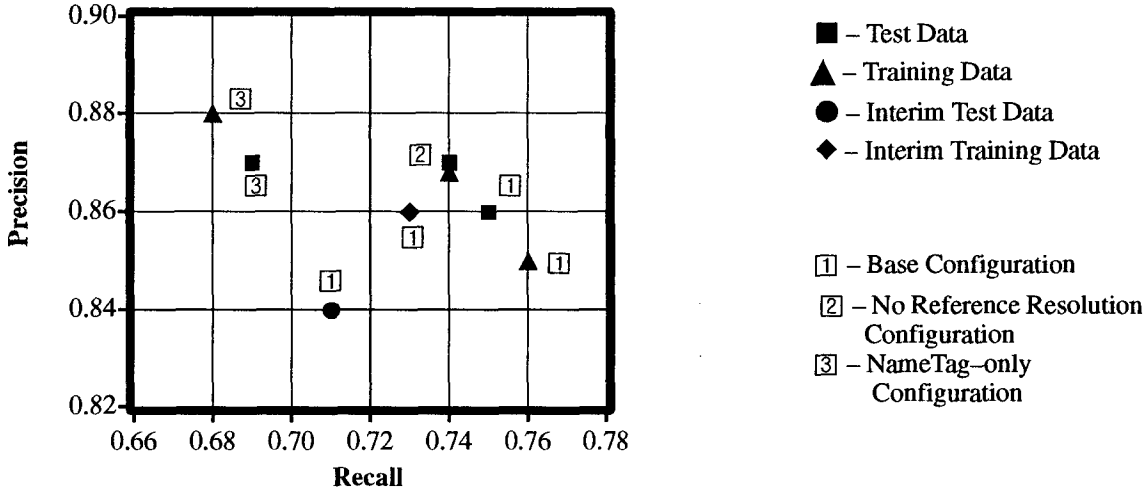

Figure 20: Template Element Test Results

Table 21 shows the processing time for the three Template Element configurations, run on a Sun SPARCstation 20. The 100 final test data documents have an original size of 261,658 characters. **NameTag** took 19 seconds to process these documents, as shown in the **CPU Time** column.

Configuration	CPU Time (seconds)	Speed (Meg/hour)	Real Time (minutes)	F–Measure
BASE	152(+19)	5.51	8.09	79.85
NO–REF	75(+19)	10.02	4.08	79.99
NAMETAG–ONLY	63(+19)	11.49	3.75	76.84

Table 21: Template Element Processing Statistics

System Walkthrough

HASTEN and **NameTag** performed very well on the selected walkthrough document, achieving a recall/precision of 76/84 for the BASE and NO–REF configurations. The "Goldman" and "J. Walter Thompson" errors described in the Named Entity system walkthrough caused a spurious ORGANIZATION and PERSON object. **NameTag** classified three organization names as OTHER instead of COMPANY, due to the lack of explicit company indicators. They were "Ammirati & Puris," "PaineWebber," and "McCann–Erickson." **NameTag** also classified "Creative Artists Agency" as GOVERNMENT due to the organizational head noun AGENCY. The NAMETAG–ONLY configuration achieved a performance of 68/82. **HASTEN** generated one of the three organization descriptors, using an Egraph to extract the appositive "the big Hollywood agency," which then enabled it to extract the locale and country fills. The **Reference Resolver** did not attempt to resolve the other two descriptors. **HASTEN** did not possess an Egraph to match "Coke headquarters in Atlanta," thus causing a missing locale and country fill.

CONCLUSION

SRA used the combination of two systems for the MUC–6 tasks: **NameTag**, a commercial software product that recognizes proper names and other key phrases in text; and **HASTEN**, an experimental text extraction system. **NameTag** demonstrated high performance at high speed for the Named Entity task, as well as advanced capabilities that provided the majority of the performance for the Template Element task. **HASTEN** demonstrated a simple, flexible design using simple training examples with minimal customization for the Scenario Template task. Experimental results demonstrated the speed of customization, the relationship between the number of examples and performance, the predicted potential performance, and performance on just the core scenario event.

In the near future, **NameTag** will continue to improve its coverage, accuracy, and speed. In addition, **NameTag** will provide foreign language versions, including French, Italian, Spanish, German and Japanese. SRA will strive to make **HASTEN** easier to customize by non–developers, while enhancing its features to improve its extraction performance. Potential areas of experimentation are automatic Egraph construction, the utilization of negative examples, the utilization of mutated Egraphs, and user–in–the–loop feedback. **HASTEN**'s modular design will also facilitate the integration of other supporting software modules such as syntactic parsers and discourse modules.

ACKNOWLEDGEMENTS

Kevin Hausman implemented all of the C++ code for the **NameTag**™ engine. Kurt Dusterhoff created training data and conducted valuable testing for **NameTag**.

Since 1986, SRA has developed NLP systems, mainly in the area of text extraction in both English and foreign languages. SRA developed a generic text understanding prototype **Solomon**, which was used for text extraction in several domains and languages, including MUC–4, MUC–5, Murasaki, and IAA. Currently, SRA is focusing on the issues of speed, robustness, portability, and trainability. **NameTag** is a side–effect of the effort to create a fast, robust and portable multilingual preprocessor, called **TurboTag**. Besides **HASTEN**, SRA has conducted other research in automatically trainable systems, including co–reference resolution for the SENSEMAKER project. SRA has also developed graphical annotation tools (e.g. **NameTool**, **Discourse Tagging Tool**) to support the creation of training data for automated acquisition. The Natural Language group is led by Chinatsu Aone, and key members include Kevin Hausman, Sharon Flank, Scott Bennett, John Maloney, Hamid Bacha, Job van Zuijlen, and Arcel Castillo.