Interactive Annotator Learner

11 – 792

Software Engineering for Information Systems

Benjamin Lambert

Jose Alavedra

Table of Contents

Abstract

When using machine learning techniques for a new NLP problem or in a new domain one of the least exciting aspects is manually labeling new training data for the new problem/domain. The current process of manually annotating text can be very time-consuming, tedious, and expensive. We show how integrating the manual annotation process with the machine learning process makes the task easier, faster, and more effective.

By integrating these two processes, manual annotation and machine learning, they can help each other. The learning algorithm can help the user by proposing new annotations to the user who simply confirms, edits, or deletes the proposed new annotations (rather than hunting through the corpus for them). Likewise, the human annotator can help the learning algorithm by annotating examples that the learning algorithm thinks will be most informative (i.e. it actively learns).

We call this integrated system an Interactive Annotator Learner because it allows the human annotator to interact directly with the machine learning process. This is as opposed to the typical procedure where manual annotation completed before the data is exposed to the learning algorithms. We have implemented an Interactive Annotator Learner that achieves this goal. We also show how this system can be used to improve the learning curve of learning algorithms and how it reduces manual labor. We do this by simulating the manual annotation process using previously annotated data.

1. Introduction

The purpose of this work is to develop a prototype of an Interactive Annotator Learner (IAL) system that shows the advantages and disadvantages of combining manual annotations[1] with machine learning processes, in contrast to a traditional manual annotation process.

The prototype has been developed as part of the 11-792 Software Engineering for Information Technology course at Carnegie Mellon University during the spring 2007 semester.

2. Original Conception

The original elements considered for the IAL are depicted in the next figure.

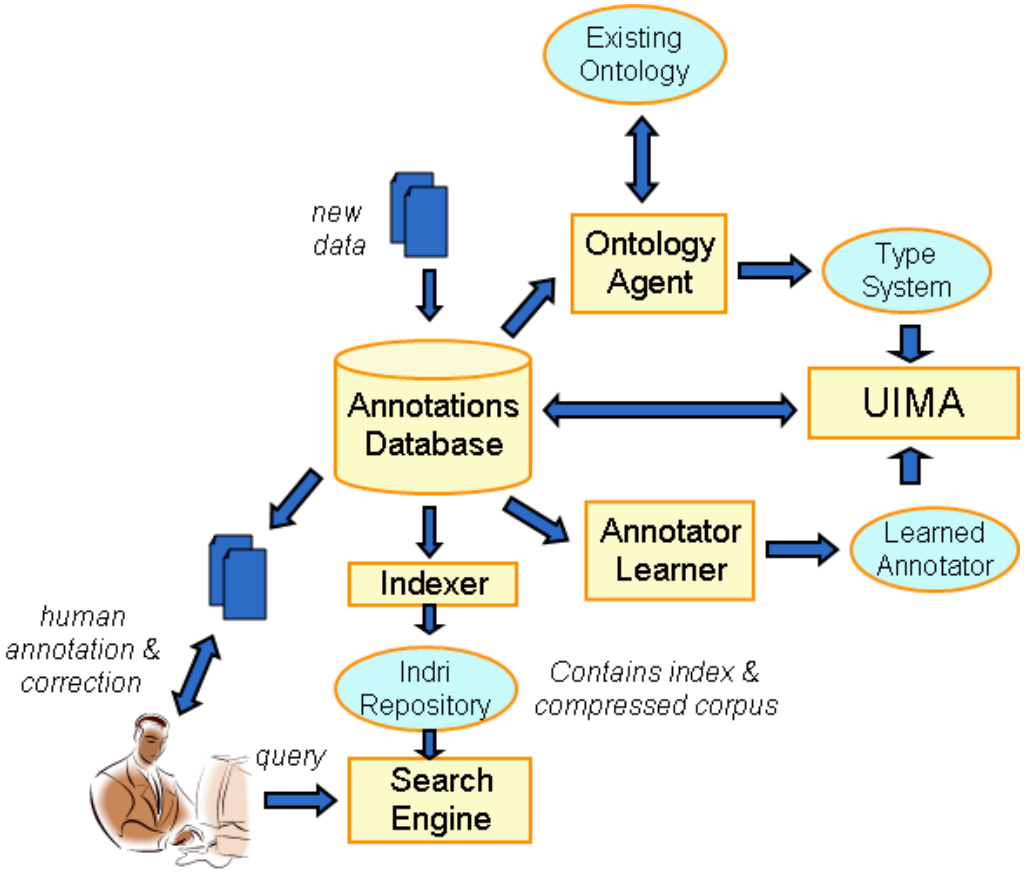


Figure 1 Interactive Annotator Learner original conception

- It shows a human querying an annotation database using a search engine.
- Based on the documents retrieved, the user can create and store new annotations in a repository (annotations database)
- An annotator learner (machine learning piece of software) can then use the set of stored annotations to train a learned annotator

[1] Please refer to the glossary of terms at then end of the document for descriptions of the concepts used in the document (e.g. annotation, annotator, type, etc)

- The learned annotator is used to automatically create new annotations in the repository that the user can then query, confirm or update
- The ontology system is used to provide a context for the annotations, defining relationships between annotation types and even hierarchies between them
- The IAL system also provides a set of recommended documents that can be used by the user to improve the accuracy of the learned annotator

3. Project Scope

The IAL system developed during the spring semester includes all the elements described in the previous section with the exception of the Ontology System. After deliberation of the functionalities that could be developed during the four months corresponding to the spring semester and in accordance with the must-have functionalities specified by the stakeholders, our team committed to develop the following modules:

- A GUI that allows the user to perform basic annotation operations in a document
- A training module to create learned annotators
- An evaluation module that permits assess the performance of the learned annotators
- A recommendation module that provides a list of documents to the user to improve the performance of the learned annotators
- A searching module that allows the user to retrieve documents from the repository based on annotation types or general words

4. Software Engineering process

4.1. Methodology

In this project we used iterative software development practices learned as part of the 11-791 Software Engineering for IS course during the fall 2006 semester.

4.1.1. Planning

- After we had a list of functionalities required for our project and we had established a preliminary scope for it, our team defined a set of milestones and activities to perform during the different project phases. We used Trac (Integrated Software Configuration Management and Project Management) as a centralized tool to follow up the activities each team member had to accomplish and to have a centralized view of the project status. In this web site we defined the milestones for our software solution, the tasks (represented as tickets in Trac) that should be accomplished as part of each milestone, and the responsible for each of these tasks. In addition, we created tickets for each issue identified during the different development phases. Even though our team was composed by two people and maintained close communication about the current issues, the use of this tool was useful to have a big picture of the status of our project, track the percentage of progress accomplished and quickly identify risks.

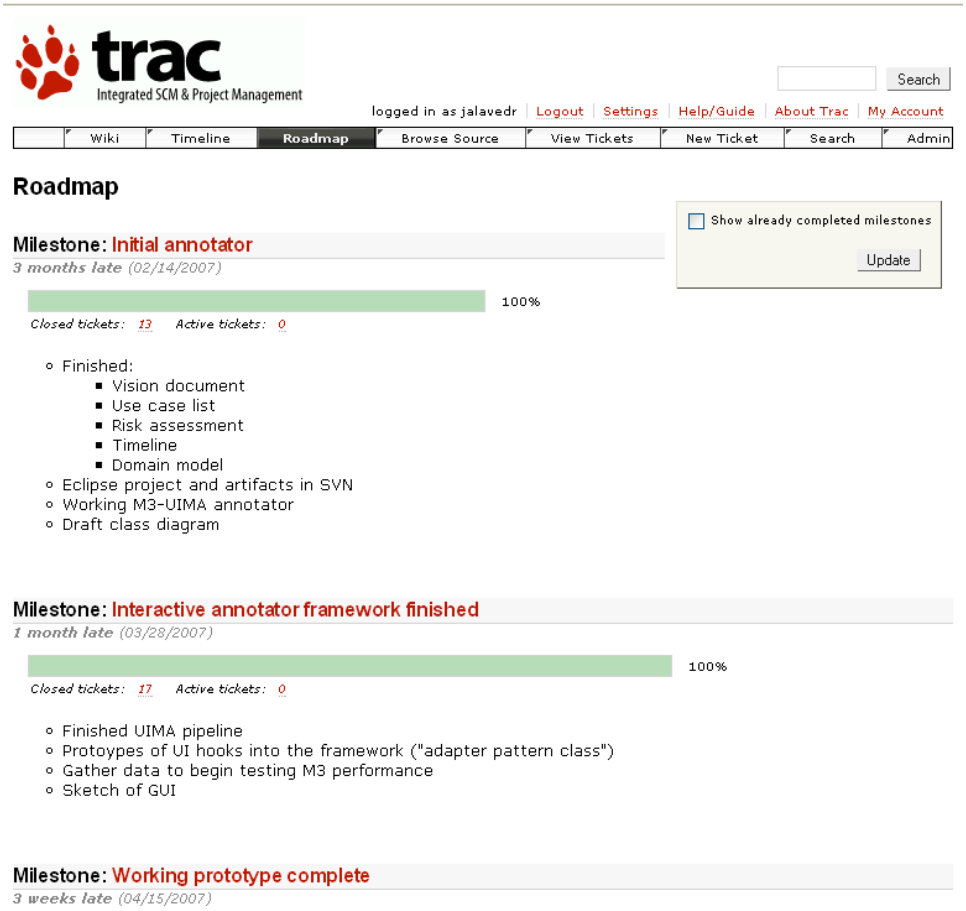
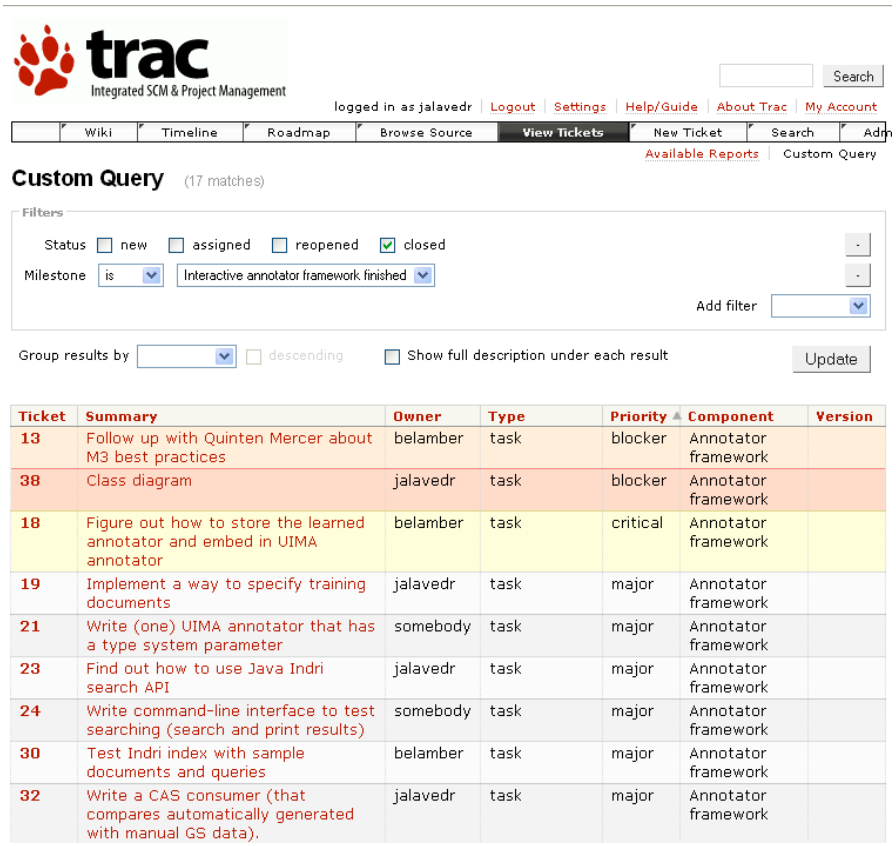


Figure 2 Milestones created in Trac



Ticket	Summary	Owner	Type	Priority	Component	Version
13	Follow up with Quinten Mercer about M3 best practices	belamber	task	blocker	Annotator framework	
38	Class diagram	jalavedr	task	blocker	Annotator framework	
18	Figure out how to store the learned annotator and embed in UIMA annotator	belamber	task	critical	Annotator framework	
19	Implement a way to specify training documents	jalavedr	task	major	Annotator framework	
21	Write (one) UIMA annotator that has a type system parameter	somebody	task	major	Annotator framework	
23	Find out how to use Java Indri search API	jalavedr	task	major	Annotator framework	
24	Write command-line interface to test searching (search and print results)	somebody	task	major	Annotator framework	
30	Test Indri index with sample documents and queries	belamber	task	major	Annotator framework	
32	Write a CAS consumer (that compares automatically generated with manual GS data).	jalavedr	task	major	Annotator framework	

Figure 3 Tickets associated to the Interactive annotator framework milestone

4.1.2. Task priorities

Our team performed a risk analysis to prioritize the use cases and activities that should be performed at the beginning of the project (those with highest risk).

The following figures show the result of this analysis.

Use Case	Task	Risk	Importance
UC6	Learn how M3 works (high)	1	1
UC6	Convert XCAS file to M3 file format	1	1
UC6	How to configure M3 - Choose learning algorithm - Choose feature selection process ???	1	1
UC6	Figure out how to store the learned annotator and embed in UIMA annotator	1	1
UC6	Implement a way to specify training documents	1	1
UC7	Convert M3 output to annotation in DB (easy? any post-processing of M3 output?)	2	2
UC7	Write (one) UIMA annotator that has a type system parameter	2	2
UC7	Automatically generate type system descriptor(s) and subclasses of UIMA's Annotation class - Generate AE descriptors (XML) ???? (may only need to update the type system descriptors) - Generate type system descriptors (XML) - Call JCasGen on type system to generate the Annotation types	2	2
UC7	Implement a way to specify which documents to annotate (same as above?)	2	2
UC2	Find out how to use Java Indri search API (Matt knows??)	3	4
UC2	Write command-line interface to test searching (search and print results)	3	4
UC2	Write GUI for searching (later; low priority)	3	4
UC3,UC4	Write a GUI for annotations 0. see if we can get source for xcasviewerapp or if Eric Riebling (or someone) can work with us. 1. use xcasviewerapp?? (doesn't have newAnnotationType() functionality) -Find out if we can extend this to add new types on the fly -Wouldn't need to change this at all if we implement (part of) UC5 and were only adding annotations, but we need to be able to "verify", edit, and delete annotations 2. reuse UIMA viewer (doesn't add annotation functionality)? 3. write something from scratch	4	3
UC5	Implement "add new type" which triggers new type system re-build	5	5
UC5	Delay "RUD" types until later.	5	5
UC10	Devise a model of user actions, and keep track of them	6	9
UC8	Install Indri CAS consumer	7	6
UC8	Test Indri index with sample documents and queries	7	6
UC8	Connect to UI	7	6
UC11	Write a CAS consumer (that compares automatically generated with manually).	8	7
UC11	Use the methods defined for choosing which documents to evaluate on	8	7
UC9	Write CAS consumer to get corpus annotation stats	9	8
UC12	Implement something trivial (recommend documents that don't have any annotations of some type or documents with lowest accuracy/P/R)	10	11
UC1	Create GUI to add/remove documents	11	10

Figure 4 Prioritized use cases and tasks

UC1. Add /Remove document(s) to/from corpus
UC2. Search for documents * (assumes corpus is indexed)
UC3. View document *
UC4. CRUD + Verify annotations*
UC5. CRUD types
UC6. Retrain the annotators (include support to indicate types)*
UC7. Run annotators on some docs*
UC8. Index corpus*
UC9. Get corpus statistics
UC10. Selective undo annotations
UC11. Evaluate trained annotator*
UC12 Recommend documents

Figure 5 IAL use cases

4.2. Requirements

4.2.1. Vision

The interactive annotator learner simplifies the task of annotating text and training automatic annotators. It combines all the tools needed for:

- annotating text documents
- creating and managing a type system

- training, running, and testing automatic annotators
- managing training and test document collections
- searching documents for text to annotate/correct

In addition, the system will be able to recommend documents that will most likely help the learning algorithms to improve their performance quickly.

The advantage of this integrated approach is that the human annotator can see exactly how well the trained automatic annotators perform and correct the automatic annotator's mistakes. A good learning algorithm will learn quickly from the user's corrections and then improve the automatic annotator to make fewer mistakes on each annotate/train/test phase. As the automatic annotator gets better and better, the user's role becomes more to verify and correct annotations rather than to create all the annotations manually.

Since some learning algorithms have proven to perform very well with little training data, this should greatly simplify the task of annotating text. This is good for all parties because the process may become very tedious for the annotator if they have no assistance with the task. Manual annotators are also expensive to train and employ, so the interactive annotator learning will reduce the labor costs of hiring annotators.

Imagined sequence of events for a typical user (annotating a building):

1. User opens document from corpus
2. User highlights "NSH," enter a type and click on "Annotate"
3. System sends annotation to ADB
4. System sends annotation to Minorthird (machine learning framework)
5. System updates the type system
6. Minorthird updates its automatic annotator
7. The system runs Minorthird annotator on entire corpus
8. User searches for his/her annotation type and fixes any mis-annotated text
    a. Adding more annotations
    b. Deleting wrong annotations
    c. Adjusting annotations (e.g. changing the span)
9. System sends corrections to Minorthird

Initial prototypes

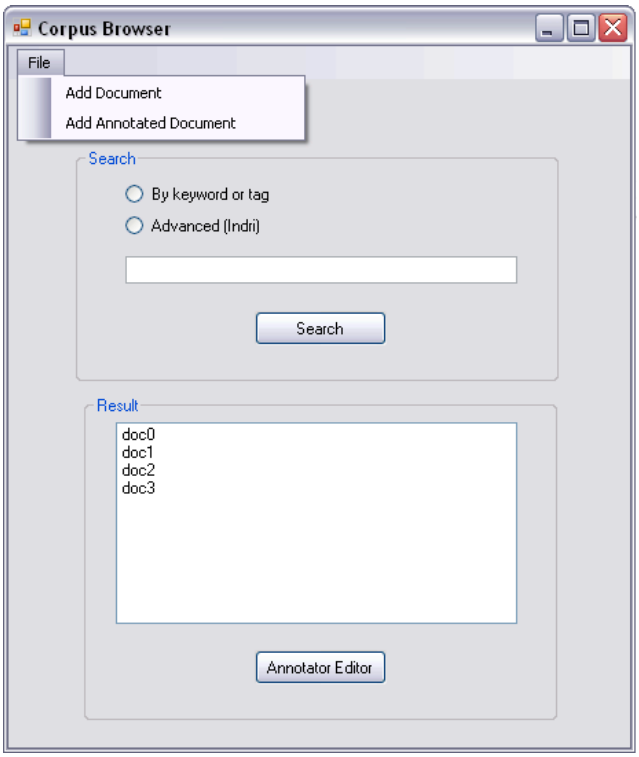


Figure 6 Search and corpus browse options

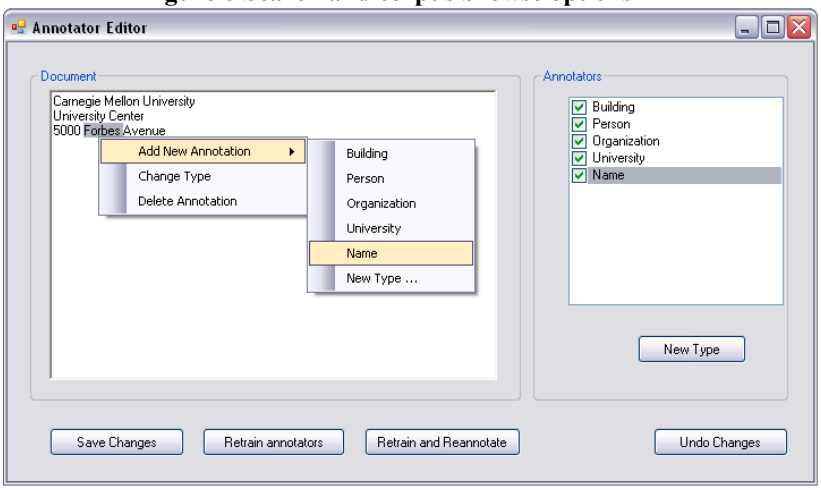


Figure 7 Annotation and Type editor

4.2.2. Use Cases

The following list shows the use cases identified during the analysis phase. Even though they are the desired functionality for the project, our team focused on developing only a subset of this use cases and some times a reduced number of features corresponding to a particular use case. A note will be included at the end of the use cases that were not implemented at all or when a simplified version was developed in the prototype.

4.2.2.1. Brief-format use cases

a. **UC1**: Add/Remove document(s) to/from corpus

In the corpus viewer, user indicates they would like to add or remove a document from the corpus. User selects the document to be added or removed. System asks user to confirm choice. Documents are copied to/from corpus repository and corpus viewer component is notified of the change.

Note: This use case was not implemented. The IAL system uses a predefined corpus or set of documents. The user could add or remove documents manually though.

b. **UC2:** Search for documents

In the corpus viewer, user enters a query in query input interface. Query consists of a combination of tags and text (including operators for overlapping tags and text, Boolean query operators, and proximity operators). System identifies files in the corpus matching the query, and displays a list of the matching documents.

c. **UC3**: View document

In the corpus viewer, user indicates which document they would like to view and/or annotate. System opens the document in the document viewer which displays the document as well as options for controlling which annotation types are to be highlighted.

Note: The GUI developed supports different annotation types in a document, but does not offer an option to filter which annotations will be highlighted.

d. **UC4**: CRUD + Verify annotations

In the document viewer, the user is presented with the text of the open document with user-specified annotations highlighted. Simply viewing the annotations is the 'read' component of this UC. The user may additionally add new annotations and 'edit' existing annotations (which may mean either adjusting the span or changing the annotation type). Finally, as a special case of the 'update' case, we allow the user to 'verify' an automatically generated annotation (this is really an update operation because it's changing an attribute of the annotation). Each of the CRUD steps registers with the system immediately and so the display updates immediately. The user must indicate that they would like to save the document for the changes to be written to permanent storage and for the changes to be available to the learning/training algorithms.

Note: The delete operation has not been developed.

e. **UC5:** CRUD types

In either the document viewer or the corpus viewer, user indicates that they would like to update the type system/ontology. The user may view the type system hierarchy and structure. The user may then change the type system by indicating which type he would like to change then he may change either the name or the parent type. The user may also delete types. All changes to the type system, especially changes to types that already have annotations in the corpus must be confirmed. Changing or a deleting any type prompts the user to specify how the

change should be reflected in the corpus (e.g. when an annotation is renamed, do we go through the corpus and rename all instances of that annotation type? When a type is deleted, do we then go through the corpus and delete all annotations of that type?)
Note: Only the create option has been implemented. The ontology system was out of the scope of the prototype.

f. UC6: Retrain the annotators

In the corpus viewer, the user selects the Retrain annotators option. The system shows a form where the user can search and select multiple types, search and select multiple documents (training data) and select the frequency for running the retraining annotator process. After the user fills the form and selects the run option, the system saves the specified options. Then, the system sends the list of types and documents to the Machine Learning Subsystem. The Machine Learning Subsystem reads the training documents from the document server subsystem and retrains the annotators. It also sends the trained annotators to the annotator subsystem. Finally, the system reports to the user the result of the process.
Note: The IAL system was intended to support synchronous and asynchronous modes to train annotators. Right now only the synchronous mode has been implemented.

g. UC7. Run annotators on some docs

In the corpus viewer, the user selects the Re-annotate option. The system shows a form where the user can search and select multiple types, search and select multiple documents (testing data), search and select multiple gold data documents and select the frequency for running the re-annotation process. After the user fills the form and selects the run option, the system saves the specified options. The system sends the list of types and documents to the Annotator subsystem. The Annotator subsystem reads and updates the specified documents from the document server subsystem. Finally, the system reports to the user the result of the process.
Note: The system allows running one type at a time. The system supports synchronous mode for running the annotators.

h. **UC8:** Index corpus

In the corpus viewer, the user can select the re-index option. The system will present a form where the user can specify the frequency of the re-index process. Then, this re-index process will read the Annotations database and populate the Indri Repository using the types available in the system as index keys. All this process will allow the user to query the documents that matches the criteria specified in a search engine.

i. **UC9:** Visualize corpus

In the corpus viewer, the user can select the Visualize corpus option. Then, the system will show statistics about the percentage of documents that have been manually annotated, automatically annotated or both.

Note: This use case has not been implemented.

j. **UC10:** Selective undo annotations

In the document viewer, the user can review the automatically generated annotations. Using the type list included in the document viewer, the user can select the types he/she considers do not have a good performance. After having selected a set of types, the user can select the Undo option. Then, the system will require a user confirmation to rollback the changes performed over the corpus by the latest automatic annotation process (UC7). The undo process only affects the annotations corresponding to the types selected by the user.
Note: This use case has not been implemented. However, each automatic annotation includes an attribute that indicates in which iteration it was created. With this feature, it is possible to implement this use case.

k. **UC11:** Evaluate trained annotator

In the corpus viewer, the user can select the "Evaluate trained annotator" option. The system will show a form in which the user can specify the location of gold-standard data or manually select a set of documents that will be used to assess the trained annotator performance. In addition, the user can specify a list of types that he/she wants to evaluate. When the user selects the run option in the loaded form, the evaluation process will begin. The system evaluates and reports the performance of the annotators taking into account the specified types and gold standard data.
Note: The location of the gold-standard data is preset in a configuration file so it is not specified by the use through the GUI. The system evaluates an annotation type at a time.

l. **UC12.** Recommend documents to annotate

User selects a type or type system that they would like to annotate. System presents a ranked list of documents that will help the system to learn the types in the type system.

4.2.2.2.Fully dressed use cases

The main use cases identified during the requirements phase were also documented in a fully dressed format

- **Use Case UC4: CRUD + Verify annotations**

Scope: Interactive Annotator Learner
Level: user goal
Primary Actor: Annotator user
Stakeholders and interests:

- Annotator user: Wants to add new annotations for the system to learn from and wants to edit, delete, and verify automatically generated annotations.
- Machine Learning Subsystem: wants to receive good training data or feedback to update its annotators.

- Document server subsystem: Wants to update its index to reflect the changes in the annotations.

Preconditions: Document viewer is displaying a document.
Success guarantees: The updated annotations are saved to a permanent storage location and are available to the learning algorithm.
Main success Scenario:

1. System displays a list of all the annotation types in the current document
2. User specifies which annotation types are to be highlighted.
3. System (or user) assigns colors or patterns for each annotation type specified and highlights each annotation of that type with the designated color/pattern. (R in CRUD)
4. To add an annotation: user selects an annotation type, then selects a span of text, and then indicates that a new annotation be created at that span. (C)
5. New annotation is recorded and the display reflects the new annotation.
6. To edit an annotation: user updates the span or the annotation type of a visible annotation. (U)
7. System asks user to confirm update
8. New annotation is registered and the display is updated
9. To delete an annotation: user selects a visible annotation and indicates that it should be deleted. (D)
10. System prompts user to confirm deletion
11. Annotation is removed from the system and the display is updated.

User repeats steps 4-11 until finished.

12. User indicates that they would like to save the changes
13. System saves all changes to the permanent storage location.
14. User closes the document viewer.

Extensions:
1a. If there are no annotations in the current document, the system indicates so.
3a. Overlapping or embedded annotations are displayed with a special emphasis so the user knows there are multiple annotations in the region.
4a. If the span or type is invalid, prompt user to retry
4b. If an identical annotation is already present, warn the user, and have them confirm that they would like to proceed.
6a. If the span or type is invalid, prompt user to retry
6b. If there is more than one annotation at the specified location, prompt user to choose which annotation they would like to edit.
9a. If there is more than one annotation at the specified location, prompt user to choose which annotation they would like to delete.
9b. If the annotation contains any non-visible information such as attributes, alert the user that the information will also be lost.
14a. If changes have not been saved prompt the user to save changes first.

Special requirements:

There should be an option to display annotations with patterns rather than colors for the color-blind.

Frequency of Occurrence:

May occur as frequently as every 10 seconds or every few minutes

Open Issues:

- **Use Case UC6: Retrain Annotators**

Scope: Interactive Annotator Learner
Level: user goal
Primary Actor: Annotator user
Stakeholders and interests:

- Annotator user: Wants to select the types and as a consequence the annotators that must be retrained. Wants to select the documents from a corpus that will be used as training data. Wants to select the frequency of the annotator training process (on demand, daily, weekly, monthly).
- Machine learning subsystem: Wants to receive a set of documents and a type system to retrain the annotators.
- Annotator subsystem: Wants to receive a set of trained annotators from the Machine Learning Subsystem.
- Document server subsystem: Wants to receive queries from the user to identify the documents (list) to be annotated. Wants to provide the documents (content) that will be used by the Machine Learning subsystem to retrain the annotators.

Preconditions: User interface is loaded, the document server and machine learning subsystems are available.
Success guarantees: The annotators selected by the user (through the type system) are retrained using the data specified. The documents used in the retraining annotator process will be updated to indicate they were used as training data. The list of types, documents and options selected by the user are saved.
Main success Scenario:

1. The user selects the Retrain annotators option.
2. The system shows a form where the user can:
    a. Search and select multiple types
    - Search and select multiple documents (training data)
    - Select the frequency for running the retraining annotator process
3. The user fills the form
4. The user selects the run option
5. The system saves the options chosen by the user.
6. The system sends the list of types and documents to the Machine Learning Subsystem.
7. The Machine Learning Subsystem reads the training documents from the document server subsystem.
8. The Machine Learning Subsystem retrains the annotators.

9. The Machine Learning Subsystem sends the trained annotators to the annotator subsystem.
10. The system reports to the user the result of the process.

Extensions:

*a. At any time system fails:

1. The user must restart the system and request recovery of prior state
2. System reconstructs prior state

2.a. System determines that the original retraining process is running.

1. System notifies the user and does not allow run the process again until the original one has finished

3a. Document server or type system not available:

1. The system notifies the user about the errors.

4a. Missing obligatory fills:

1.The user could omit one of the mandatory fields, for example "training data". In this case the system shows an error message and the user must complete the required fields.

Special requirements:

None

Technology and Data Variations List:

The system should provide asynchronous notifications when the retraining process finishes.

Frequency of Occurrence:

Nearly continuous.

Open Issues:

Precise process frequencies that should be supported by the system.

- **Use Case UC7: Run trained Annotators**

Scope: Interactive Annotator Learner
Level: user goal
Primary Actor: Annotator user
Stakeholders and interests:

- Annotator user: Wants to select the types and as a consequence the annotators that must be run over a set of documents. Wants to select the documents from a corpus that will be used as testing data and evaluate the performance of the trained annotators. Wants to select the frequency to run the annotators over a set of documents (on demand, daily, weekly, monthly). Wants also options to run the annotators over a set of documents in a corpus to create gold data annotations.
- Annotator subsystem: Wants to receive a set of documents and a list of types to determine the annotators that must be run.
- Document server subsystem: Wants to provide the documents that must be updated by the annotator subsystem.

Preconditions: User interface is loaded, the document server and annotator subsystems are available.
Success guarantees: The annotators selected by the user (through the type system) are run over the data specified. The documents used in the testing process will contain new annotations created by the trained annotators and will include a field to indicate they were used as testing data.
Main success Scenario:

1. The user selects the Re-annotate option.
2. The system shows a form where the user can:
    - Search and select multiple types
    - Search and select multiple documents (testing data)
    - Search and select multiple gold data documents
    - Select the frequency for running the re-annotation process
3. The user fills the form
4. The user selects the run option
5. The system saves the options chosen by the user.
6. The system sends the list of types and documents to the Annotator subsystem.
7. The Annotator subsystem reads and updates the specified documents from the document server subsystem.
8. The system reports to the user the result of the process.

Extensions:
*a. At any time system fails:
1. The user must restart the system and request recovery of prior state
2. System reconstructs prior state
2.a. System determines that the original re-annotation process is running.
1. System notifies the user and does not allow run the process again until the original one has finished
3a. Document server or type system not available:
1. The system notifies the user about the errors.
4a. Missing obligatory fills:
1.The user could omit one of the mandatory fields, for example “training data”. In this case the system shows an error message and the user must complete the required fields.
Special requirements:
None
Technology and Data Variations List:
The system should provide asynchronous notifications when the re-annotation process finishes.
Frequency of Occurrence:
Nearly continuous.
Open Issues:
Precise process frequencies that should be supported by the system.

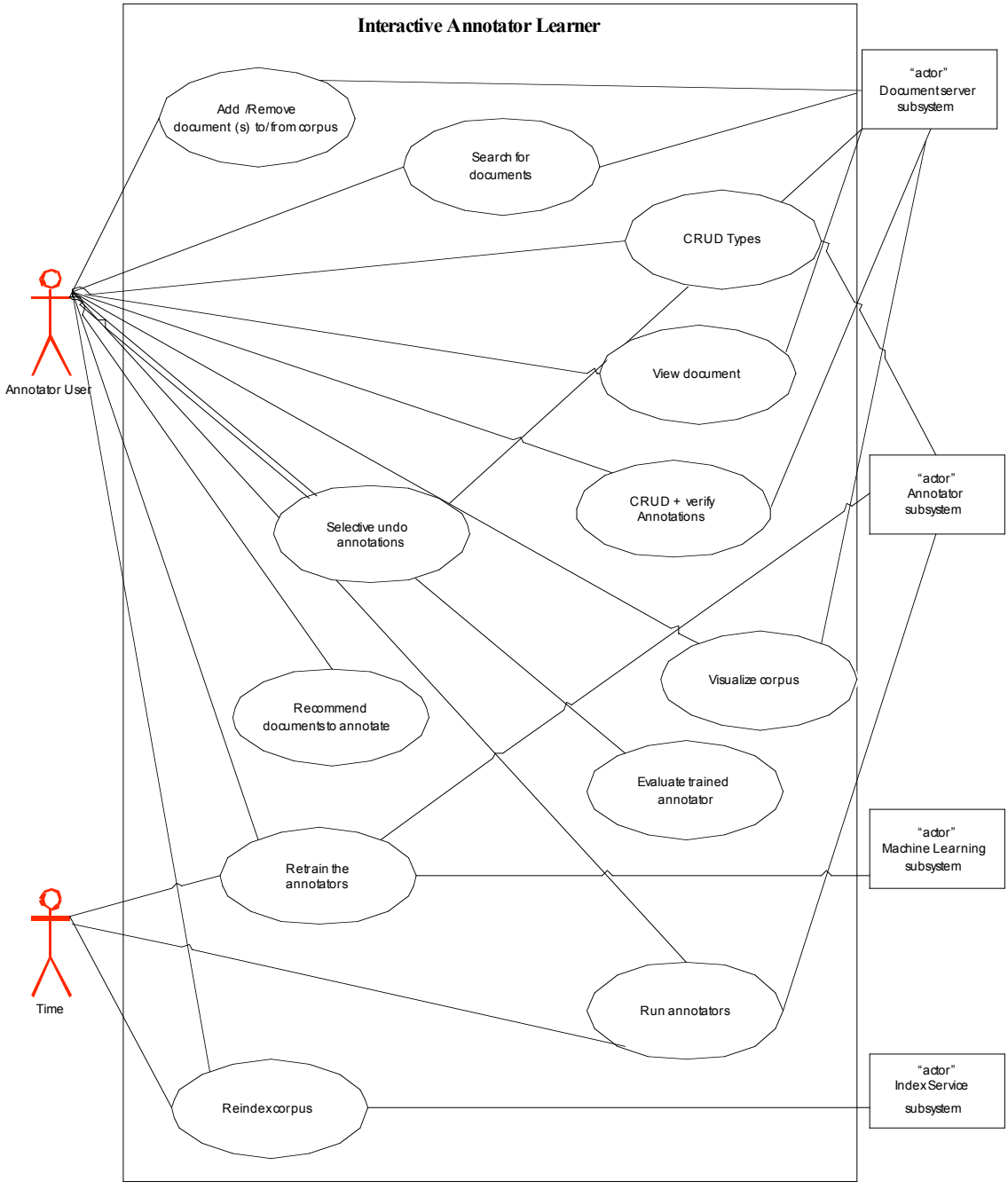


Figure 8 Use Case Diagram

4.3. Analysis

4.3.1.1.Domain Model

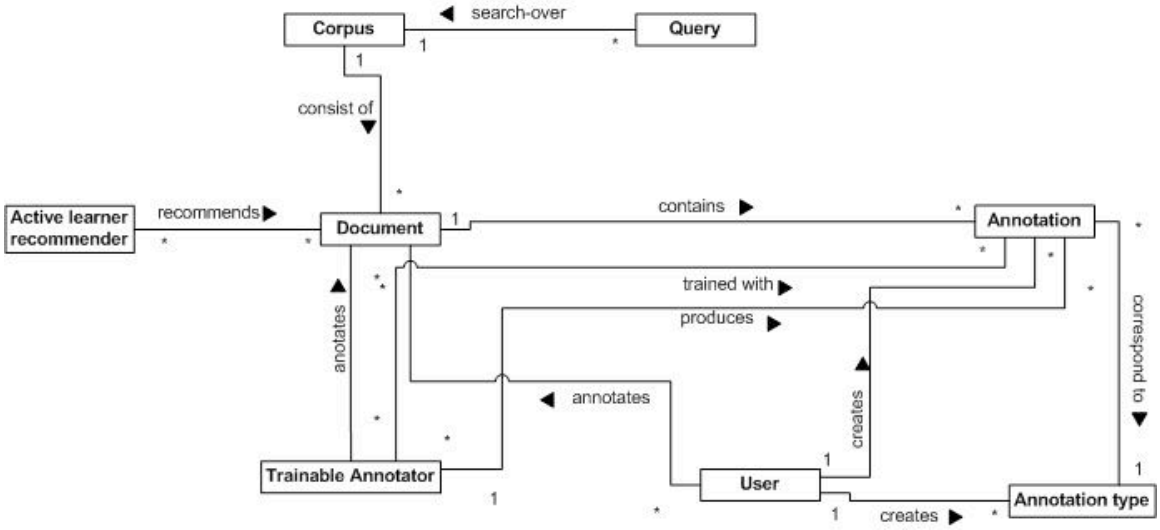


Figure 9 Domain Model

4.4. Design

4.4.1. Architecture

The following figure shows the different layers envisioned for our application (even though it can be different from our prototype). While the domain layer contains the main components of our system, the techinical services include the frameworks and packages that we used in the implementation. Our technical vision of an IAL system considers that the technical services can be easily replaced with other components in the future.

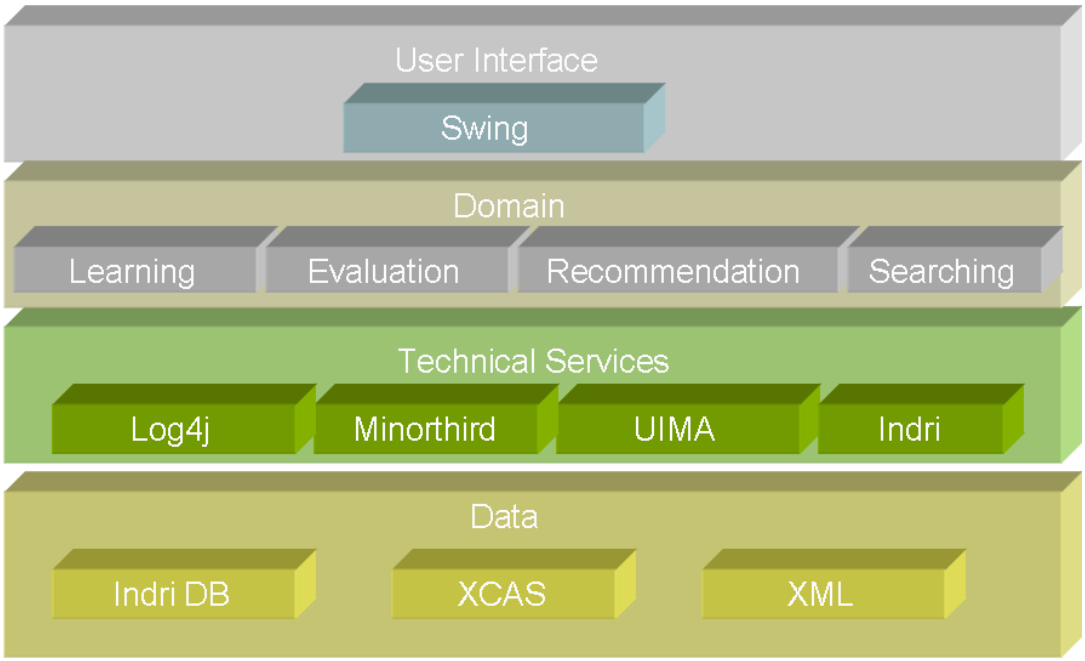
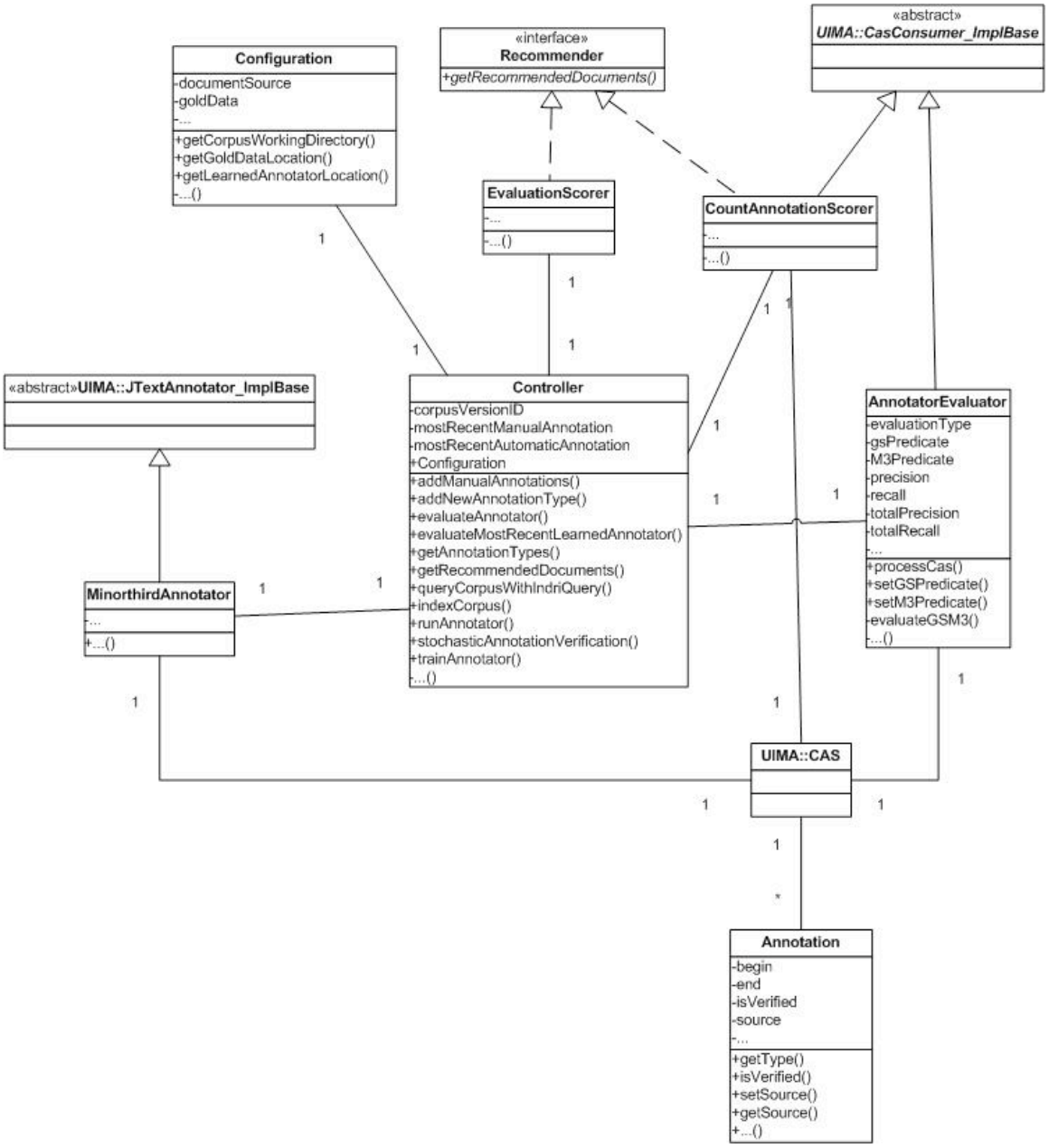


Figure 10 Logical Architecture

4.4.2. Design Class Diagram



4.5. Implementation

4.5.1. GUI

The IAL system provides a basic GUI that includes the following options:

- Add new types – Allows the user to create arbitrary annotation types
- Add new annotations – Add new annotations to a document based on the types created by the user

- Train annotators – This option uses Minorthird's machine learning algorithms to train a new annotator. The system ask the user to specify a set of documents for training
- Run annotators – Once an annotator has been trained, the user can specify a set of documents for testing.
- Evaluate annotators – This option evaluates the performance of the learned annotator calculating precision and recall statistics for each individual document used during the testing process and overall statistics as well.
- Index corpus – Start an indexing process that allows the user to search for documents using different criteria (keywords or indri sentences for instance)
- Search – Uses the index created with the Index corpus option to retrieve documents that the user wants to annotate
- Get recommended documents – Retrieves a set of documents that the IAL system recommends to improve the accuracy of a learned annotator. The system ask the user to select an annotation type and a recommendation method.

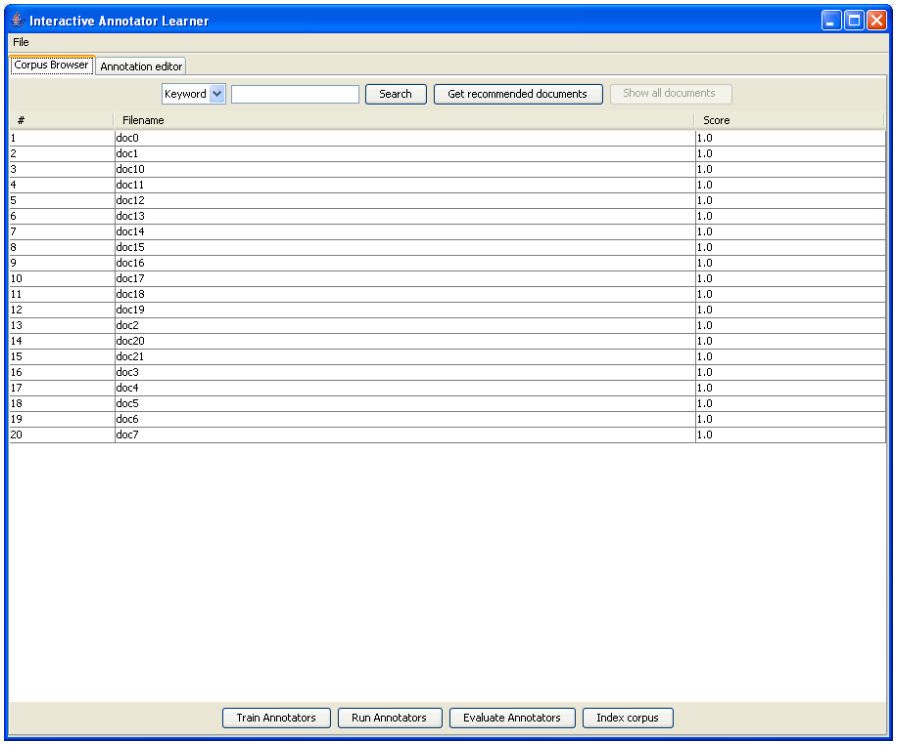


Figure 11 IAL main options

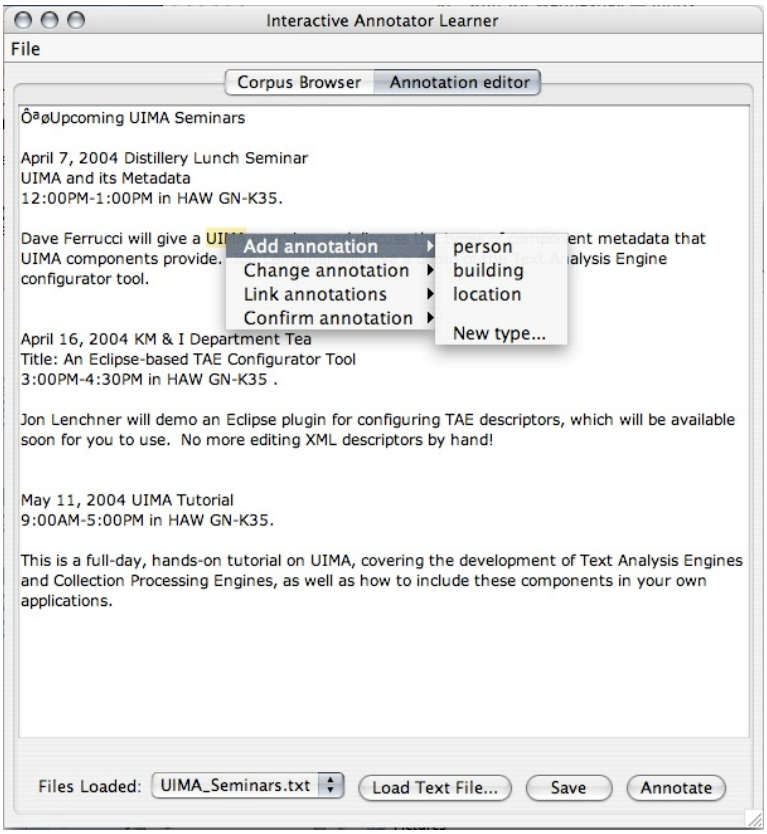


Figure 12 Adding annotations

4.5.2. Evaluation

The following statistics were captured to assess the performance of learned annotators:

- Precision = #auto-annotator got right / #it guessed
- Recall = #learned annotator got right / #it should have gotten right
- F-measure an average of P and R

These statistics are recorded in an XML file that then is used in the recommendation option.

4.5.3. Document recommendations

As mentioned before, the IAL system allows the user to select the recommendation method that will be used to obtain a set of documents to improve a learned annotator. Our team included this feature since there is not evidence that a single recommendation method can be the best option in all the cases.

- Lowest precision and recall average
- Highest precision and recall average
- Lowest precision
- Highest precision
- Lowest recall
- Highest recall

- Lowest F1
- Highest F1
- Most manual annotations
- Fewest manual annotations

4.6. Results

4.6.1. Experiment goals

- Interactive annotating reduces labor ("helps people")
- Interactive annotating accelerates the learning curve of the learning algorithm ("helps computers")

The approach used to verify the advantages offered by an IAL system are:

- Quantifying the user's effort as a function of annotator performance
- Evaluating intermediate learned annotator's performance as a function of the amount of training data.

4.6.2. Evaluation metrics

- For the automatic annotator
  Precision = #auto-annotator got right / #it guessed
  Recall = #learned annotator got right / #it should have gotten right
  F-measure an average of P and R
- Human 'effort' cost function
  5¢ to add an annotation
  3¢ to fix
  1¢ to confirm

4.6.3. Human Annotator Simulator

In order to perform the experiments and simulate a real-world situation, our team needed to annotate documents manually. Since there were only two people participating in this project and because of time constraints, our team developed an annotator simulator that mimic a human behavior by using already defined gold-standard data and then creating annotations based on this information (with some mistakes to simulate an imperfect human). The simulator takes several parameters to simulate varied degrees of human annotation performance.

4.6.4. Simulation parameters

The following process simulates a person annotating type ***T***.

1. Gets the top ***n*** recommended documents for type ***T***
2. Confirms correct auto-annotations with probability ***C %***
3. Fixes auto-annotations if they are within ***k*** characters of correct with probability ***E %***.

4. Looks at the rest of the recommended documents and adds new annotations with probability ***A %***.
5. If we have not reached a plateau:
    1. Train a new automatic annotator
    2. Run the new annotator on the data
    3. Go to #1.

Note:
The simulated human doesn't ever confirm or add incorrect annotations (Good Avenue for future work).

4.6.5. Simulated case

The annotation type we try to learn is "noun phrases" (NPs). The data corresponds to the Penn TreeBank corpus. Examples:

"it"
"The economy's temperature"
"this era of frantic competition for ad dollars"
"this era"

Human parameters used in the simulation

- Batch size, N = 2 (# of recommended docs that are annotated each round)
- Confirmation probability *C %* (generally very high, 90% or 95%)
- Edit annotation threshold, $k = 6$ characters
- Edit annotation probability, *E %*. (generally lower than C%, maybe 50%)
- Add annotation probability *A %*. (variable 60%, 70%, 80%, etc)
- Semi-CRF annotator learner (learning algorithm used in the simulation)

4.6.6. Varied human performance comparison

The following figure shows the performance (F1 measure) of a simulated human adding annotations with 0.6, 0.8, 0.9 and 1 probability (there are not editions nor confirmations). As expected, the higher the probability of annotating, the higher the performance.
Based on these results, we selected a simulated human with 0.9 probabilities of adding an annotation (imperfect but even good performance) to perform further experiments.

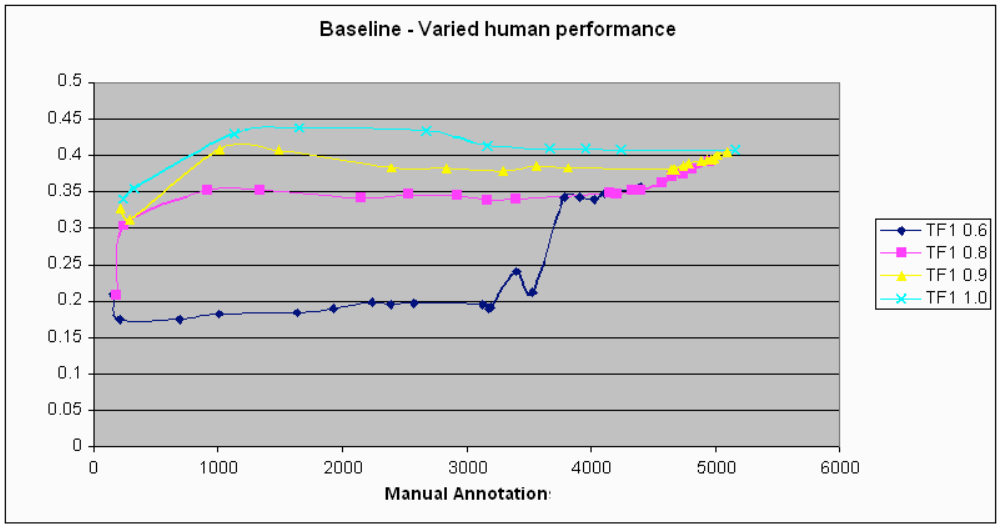


Figure 13 Varied human performance

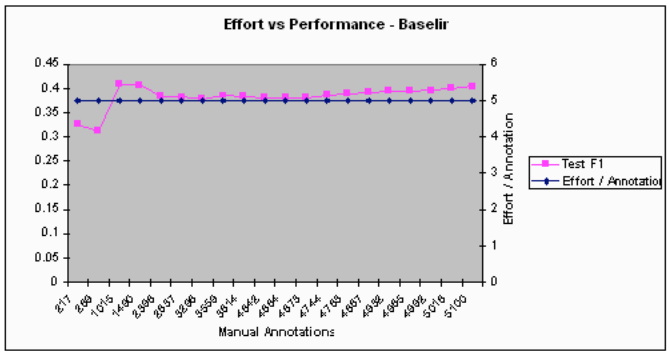
4.6.7. Decreased labor

The first figure shows the performance of a simulated human with 0.9 probabilities of adding an annotation (5¢ cost) and the effort/annotation during the process.

The second graph represents the same human but this time the IAL has created some automatic annotations so the human can edit and verify these annotations which have 3¢ and 1¢ cost respectively. As we see, the effort/annotation (blue line) is reduced compared to the previous scenario.

Note: The graphs presented in the following pages use two Y axis. The values on the left represent the F1 measure while the right side represents the effort / annotation.

N = 2
A = 90%
E = 0%
C = 0%



N = 2
A = 90%
E = 90%
C = 90%

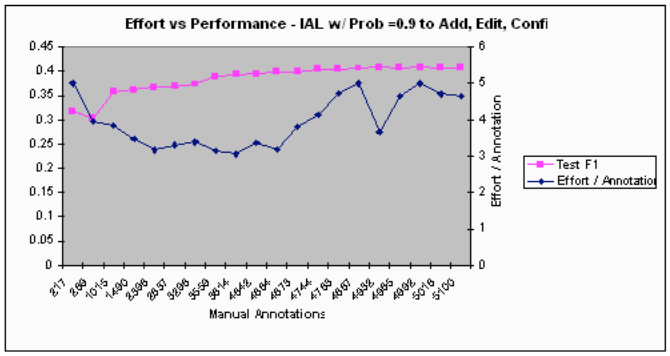
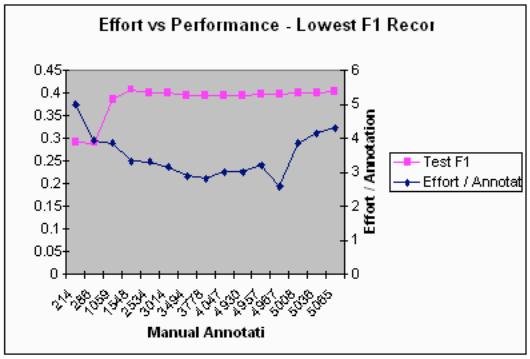
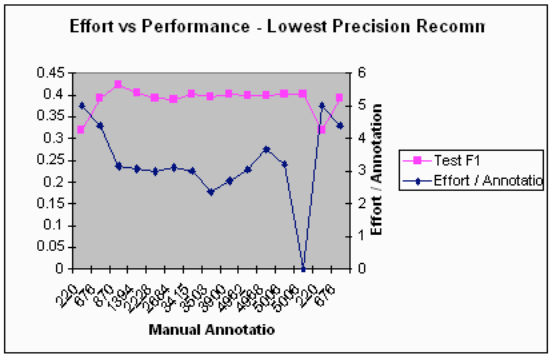


Figure 14 Effort/Annotation comparison

4.6.8. Recommenders and learning curves

The following experiments the effort/annotation corresponding to different recommendation methods. The graph on the left uses the "Lowest F1 recommender" while the right one uses the "Lowest precision recommender". In this case the graphs don't show significant advantage of one recommender over the other.





n = 2
A = 90%
E = 50%
C = 95%

Figure 15 Recommenders comparison

4.7. Conclusions

- According to the obtained results, an IAL system seems to reduce the labor required from a human annotator to achieve a performance comparable to that of the traditional manual process.
- More experiments should be performed in order to identify the best recommendation method (if it exists) and even more methods could be incorporated into the system.
- Performing some experiments can demand considerable time (more than 4 hours). Even an individual training process (a single annotator) can take some minutes, so it would be convenient consider asynchronous processes that allows the user to keep working while the most heavy processes run in the background
- The GUI has an important role in an IAL system since it can affect the effort required by a human annotator to create or update annotations. A poor design could minimize the benefits obtained by using an IAL system.

4.8. Future work

- Experiments with real humans should be performed in order to evaluate more accurately the improvement obtained with an IAL system. It would be interesting determine the real cost of adding, editing or confirming annotations and also identify other possible variables involved in a manual process such as fatigue, motivation, etc.
- Experiments that consider negative examples as part of the annotator learning process should be performed to identify if the learned annotator performance can be improved with this additional information
- Use POS tag features to compare with state of the art NP annotator learning
- Study how active learning techniques affect the learning curve (e.g. recommend documents/annotations that have low confidence annotations from Minorthird
- Examine how using an ontology of types can inform the learning (e.g. Canada is tagged as a country, so it must also be a location).

5. Reflections and lessons learned

- Working in a two people team has advantages and disadvantages. On one hand, it was easy to coordinate and keep tracking of each other progress and also was easy allocate time during the week to perform periodical meetings (we had a meeting every week). On the other hand, our project was significantly affected when even one person had a very tight schedule during some weeks due to other courses. A better planning could have been performed based on the work load of the team members.
- We didn't use detailed minutes to keep tracking of the meeting agreements. Instead of that we included the agreements directly in an email at the end of the meetings and scanned any diagram or note that was important. It in fact allowed the team members to be focused on more critical activities.
- Risk analysis is a very good practice to determine the first tasks to work on. In our case know how to use Minorthird was critical since it was the main

component of the annotation learning process. Prioritizing the risky tasks at the beginning allowed us to spend enough time to mitigate the risks and plan the other tasks with more confidence.

- Due to the short time available during the semester to complete all the tasks in the project, we spend most of the time coding new functionalities but it was difficult to allocate time for testing. Because of that, some bugs were identified almost at the end of the project so it was difficult to fix them at the last moment. We could have created a test plan at the beginning of the project to avoid these situations (it could have reduced the scope of the project though).
- Some design patterns were difficult to implement given the frameworks that we were using for the development. Because we developed a prototype and due to the tight schedule that we had, we considered that it was acceptable to simplify our design. However, in a real application, a very careful analysis of the development tools and frameworks should be done to avoid the described situation.

6. Deliverable

6.1. Acknowledge

The following classes and packages were developed in other projects and our team used them in coordination with the authors.

- edu.cmu.lti.ial.gui
- edu.cmu.lti.ial.learning.MinorthirdAnnotator
- edu.cmu.lti.ial.learning.MinorthirdCASConsumer

6.2. IAL Source code and documentation

- The source code, documentation, java API and artifacts for our project can be found in the subversion library defined for the 11-792 course.

svn://seit1.lti.cs.cmu.edu/ial07/InteractiveAnnotatorLearner

6.3. IAL Configuration

See the "IALSetupGuide.doc" file for instructions about how to configure the IAL system.

7. Glossary

- **Annotation:** The association of a label with a region of text.
- **Corpus:** A set of documents corresponding to a particular topic or area of interest
- **Document:** A text file that can be annotated
- **F1 or F-measure**: Measure of the performance of a learned annotator. Calculated as an average of Precision and Recall
- **GUI:** Stands for Graphical User Interface
- **IAL:** Interactive Annotator Learner
- **Minorthird:** Machine Learning Framework used by the IAL system to train annotators

- **Precision:** Measure of the performance of a learned annotator.
  - #auto-annotator got right / #it guessed
- **Recall:** Measure of the performance of a learned annotator.
  - #learned annotator got right / #it should have gotten right
- **Recommendation Method:** Defines the algorithm and input variables used to recommend documents
- **Scorer:** Class used by the IAL system to assign a numerical value or priority to the recommended documents. The algorithm used for each scorer is based on a recommendation method.
- **Type:** An object used to store the results of analysis. A type is roughly equivalent to a class in an object oriented programming language, or a table in a database.
- **UIMA:** Unstructured Information Management Architecture: a software architecture which specifies component interfaces, design patterns and development roles for creating, describing, discovering, composing and deploying multi-modal analysis capabilities.

Notes:

- Some terms are defined in the context of the present document.
- Some definitions were extracted from the file UIMA_SDK_Users_Guide_Reference.pdf (chapter 19)

8. References

- Fei Sha and Fernando Pereira. Shallow parsing with conditional random fields. Technical Report CIS TR MS-CIS-02-35, University of Pennsylvania, 2003.
- Sunita Sarawagi and William W. Cohen. Semi-markov conditional random fields for information extraction. In NIPS, 2004.