An Automatic Feedback System for Academic Writing

Rafael A. Calvo, Jorge Villalon, Stephen O’Rourke
The University of Sydney

Abstract
Writing is an important activity in academic and corporate environments. Feedback on the writing is key to improving the quality of documents, and for novice writers to develop their skills. This paper describes the architecture of a web-based framework for providing automated feedback. The framework allows writers to use different writing tools such as Wiki’s, cloud-based Google Docs or Word. The documents are then processed to produce a wide range of types of feedback spanning collaborative or individual writing activities, visual and text modalities, feedback can be on surface or content features, on the writing product (the final document) or the process. This range of possibilities requires an architecture that is highly flexible and extensible. The paper also provides a taxonomy of automated feedback systems that shows how other systems approached different aspects of writing support. The evaluation provides evidence on the extensibility of the framework and how it is used on realistic teaching scenarios.

This is an internal report released for those using glosser. The system can be downloaded from http://www.glosserproject.org

Keywords: Writing support, Learning technologies, Natural Language Processing, Cloud computing, Collaboration

1. Introduction
Writing is an important skill in most professional and academic endeavors. It is at the core of many learning activities and is widely recognized (e.g., Emig, 1977) for being useful developing numerous skills beyond subject matter learning where writing requires deep cognitive engagement with content. Regrettably, teaching of writing skills is difficult, expensive and often requires specialized skills that are not available. This has triggered a great deal of research on computer-based approaches for supporting writing. A substantial
part has been on techniques for Automated Essay Assessment and has mostly focused on the process of marking specific writing activities (e.g. GRE questions) automatically. Other tools have been developed with a greater focus on generating feedback that the student can use to improve their assignment. The feedback produced by these tools is of different kinds and/or focuses on different types of writing (e.g. different genres or individual rather than collaborative).

These wide-ranging types and purposes for feedback on writing means that software in this domain would benefit by being built using software engineering principles, particularly related to reusability of design and implementation (Fayad & Schmidt, 1997). The architectures of Object Oriented Application Frameworks allow developers to increase the amount of code reused by using different OO approaches (e.g. interfaces) and by implementing the common set of functionalities that a number of tools would use. This is a significant approach to having multiple systems running independently with little or no integration, as it currently happens with writing feedback deployments.

The features of writing tools and the requirements of feedback systems also keep changing and engineers find it challenging to keep systems up to date with them. Most recently, new cloud-based writing systems such as Google Docs are opening the possibility for new types of feedback, particularly those focused on the process of writing rather than just the product (i.e. the final version submitted). The need for adaptability means that the framework needs to be modular (so a component or tool can be updated without affecting the others) and extensible so new tools, not even conceived at the time of designing the framework, can be built.

Other aspects do not change as quickly, for example, collaborative writing processes or the patterns behind common teaching activities. These patterns have been studied by applied linguistics, writing and educational researchers and but are not always known in other communities. But most software developers would not be aware of them so software architectures that implement these educational patterns bring best practices to implementation (Rafael A. Calvo & Turani, 2009).

There are yet other aspects of feedback (human or automated) that we do not know much about. For example, although different writers respond with positive behavior to different types of feedback, the impact of different forms of feedback is not well understood. These behaviors may include the obvious fixing of the problem raised in the feedback, but also the learning something that can be transferred to another assignment (the most desired outcome) or a plain disregard for the feedback (the least desired outcome). Some writers may respond to
visual representations and other to text, or in fact to multiple representations that have shown to be useful in many learning situations (Ainsworth, 2006). Targeting writers with all these different forms of feedback requires either one for each form of feedback, or a framework that allows all the forms of feedback to be integrated.

Feedback should also be personalized. Prior experiences are particularly influential in the way a student engages in a writing activity, they affect not only what they know but also their motivation and engagement. Studies exploring students’ conceptions of automated feedback have shown how qualitatively different conceptions lead to different ways of using the tool and end with different learning outcomes (i.e. grades) (R.A. Calvo & Ellis, 2010). Students with different motivational drivers or learning styles would also require different type of feedback. This pedagogical requirement brings technical challenges. Having multiple systems would mean integrating the personalization module to each of them. Even in the unlikely case that it was possible (most of the systems would be proprietary and with no APIs) it would be highly inefficient.

Furthermore, most forms of feedback require a common set of document management, Natural Language Processing and machine learning techniques. Having multiple systems would mean reproducing many of these functionalities with a significant reduction in efficiency. For example, when a student revises an assignment (or when he requests feedback) each feedback module may have to index the new revision.

In this paper we describe Glosser, a web-based application framework that addresses the challenges above: 1) Glosser provides multiple forms of automated feedback 2) The architecture addresses the challenges of the collaborative and individual writing situations described above, 3) It has feedback that focuses on an outcome (i.e. a specific revision of the document) or on the writing process. 4) Feedback tools include those that help writers with surface features (e.g. spelling and grammar) or content (e.g. flow and topics). 5) They also include visual and text-based representations.

A number of feedback tools have been built using the framework. The tools described include two for helping writers reflect on flow, one using interactive text and another a map (O’Rourke, Calvo, & McNamara, 2011); three tools support reflection on the topics and concepts, two visual representations using different computational approaches and a textual one; one tool is for reflecting on the social and time related aspects (i.e. the process) and one generates questions automatically (Liu, Calvo, & Rus, 2010).

This paper focuses on the technical challenges of building the framework and tools, so Section 2 describes the literature on automated feedback and assessment of writing, but
focusing on the technical challenges raised by the different forms of writing and the technical and pedagogical approaches. Section 3 describes Glosser’s software architecture and a set of sample modules, each chosen to describe how different technical challenges can be best addressed. A spelling module is the simplest and shows how a feedback system can benefit from cloud based document management functionalities. A topic detection module that provides summaries of each document shows different approaches to building visual or text representations. A concept map extraction tool highlights the scalability challenges faced by the NLP components. Finally the process analysis tools shows how tens of thousands of revisions for each single document can be used to produce automated feedback on the writing processes followed by groups or individuals. Section 4 describes two types of evaluations, one of the technical aspects of the system (e.g. scalability) and another of the pedagogical aspects that inform this type of development. Section 5 discusses the challenges of using such systems in teaching environments with hundreds of students and provides insights aimed at researchers and developers interested in building architectures that, in general, add value (e.g. through feedback) to text-based content that users create and maintain on the cloud.

2. Background

A principle behind the Glosser’s design is that feedback can be any type of support that a system provides to a writer so the writer (individually or as part of a team) can revise the composition (based on the definition of feedback by Keh, 1990). Such design principle means taking into account the different pedagogical perspectives that writing instruction researchers have developed on feedback on writing and the way students respond to this feedback (Haswell, 2006; Hounsell, 1987; Weaver, 2006). Most researchers agree that writing requires the coordination of multiple perspectives (content and audience) and the linearization of thought, which might not be linear (J.R. Hayes & Flower, 1980). From a discourse theory perspective, students must learn to understand and reproduce the written discourse of their professional community's if they are to become part of it (Gee, 2004). And pedagogy-based arguments for the value of writing assert that writing is an important medium for reflection and, in the context of higher education, also a medium for developing epistemic orientations (Prain, 2006).

The writing activities can be varied and each type requires a different form of feedback. The activity may require writing to be done individually or by a group, it may be done in stages or as a single deliverable. Although most research on real (i.e. not in the laboratory) writing instruction has focused on the outcome of writing (the document
submitted) rather than the process, current technologies allow highly scalable recording of the writing process. These many factors mean that reviewing such wide literature is not possible in the scope of this paper. We aim only to provide a high level taxonomy that allows software developers to take into account different types of requirements.

Several researchers, working on instructor generated feedback, have produced taxonomies of the components and features that can be included in such feedback and they can provide examples of the design requirements for automated systems. Notably there is no consistent evidence that labor-intensive features necessarily lead to learning gains (Haswell, 2006; Shute, 2008).

2.1 Quality features in writing

There are many difficulties in providing effective feedback in writing, even more so through using computational approaches. Among the most significant is the difficulty in identifying reliable and useful textual features of good writing that can be implemented as a computer model. These features can be grouped in those that tackle the surface of the composition such as spelling and grammar, while other deep features refer to what lies below the communication aspects, that is, the actual ideas being communicated.

Early systems like Writers Workshop, developed by Bell Laboratories, and Editor (Thiesmeyer & Thiesmeyer, 1990) focused on grammar and style, features that are now easy to detect automatically, but they showed with limited pedagogical benefits (Beals, 1998). While these automated systems faced problems of accuracy, the main difficulty was not technical but the fact that correcting surface features of student texts did not help them to improve the quality of their ideas or the knowledge of the topics that they were studying.

SaK, a more recent writing tutoring system developed at the University of Memphis (Wiemer-Hastings & Graesser, 2000), is based on Flower’s (1994) notion of voices that speak to the writer during the process of composition. SaK uses avatars to give the impression of giving each of these voices a face and personality (Wiemer-Hastings & Graesser, 2000). Each avatar provides feedback on a different aspect of the composition, identifying strengths and weaknesses in the text but without offering corrections. Glosser adopts a less explicit form of Flower’s idea, using tabs rather than avatars, and providing a set of questions that scaffolds the reflection process.

SaK, like Glosser, uses Latent Semantic Analysis (LSA) to calculate the average distance between consecutive sentences and to provide feedback on the overall coherence of
the text. LSA is a technique used to measure the semantic similarity between texts and has been described thoroughly elsewhere (T. K. Landauer, D. S. McNamara, S. Dennis, & W. Kintsch, 2007). SaK can also analyze the topic of a sentence, identifying clusters of topics so when a new topic arises the student can be asked for an explanation or reformulation.

Despite having evaluated an increasing number of features, a challenging research question is still to identify one or more that are unambiguously representative of good writing (McNamara, Crossley, & McCarthy, 2010). For example, many authors assume that cohesion (or coherence) is a feature that identifies good writing. This seems to be supported by some evidence (Witte & Faigley, 1981), however, highlighting the difficulty of identifying unequivocal quality features of texts, even this commonly held assumption has been questioned in the McNamara et al. study (McNamara, et al., 2010) that showed no significant correlations between cohesion measures and essay ratings.

The purpose for the writing activity should be taken into account in the design of writing feedback systems. For example, they can focus on drills, such as Summary-Street (Wade-Stein & Kintsch, 2004) a system also based on LSA that provides feedback on students’ summaries. Their success tends to be skills-based, such as learning to summarize, rather than driven by disciplinary concepts.

2.2 Collaborative and Individual Activities

It is important for a generic feedback framework to provide functionalities for individual and for collaborative writing activities. Although most feedback and assessment systems reviewed here focus on individual writing, writing in professional practice is generally done collaboratively. Ede & Lunsford (1992), for example, show that 85% of the documents produced in offices and universities have at least two authors, there is a clear need for developing students’ collaborative writing skills.

Social awareness as a form of supporting writing teams started to be studied in the early 90’s (Dourish & Bellotti). Some social awareness tools are now generally embedded in collaborative writing tools like Google Docs, but the right way of using this information to improve collaboration is still unknown. Part of the reason is that writing patterns have not changed much over time (Lowry, Curtis, & Lowry, 2004) so despite the availability of tools for synchronous collaborative writing people, particularly novice writers, do not always value their functionality.
2.3 Representation
Multiple representations can be very helpful for students learning complex new ideas (Ainsworth, 2006), such as the ones often developed through writing activities. Researchers have studied how eternal representations reduce the effort required to solve equivalent problems (computational offloading) and how representations with the same abstract structure influence problem solving in different ways (re-representation). Each representation requires effort to understand as students need to learn how information is encoded and what it means to their writing tasks. Different representations can support reflection on different aspects of the writing. A tabular description of the topics developed in an essay would be more detail and possible express ambiguity in a way that a visual representation of the topics cannot easily convey. On the other hand, the visual representation can more easily express abstract and high level relationships.

2.4 Outcome vs. Process
Most writing feedback (and obviously assessment) focuses on the product of writing rather than on the process. In the classroom, students and instructors are accustomed to receiving feedback on the assignment that was submitted, but not often on what the student did in order to get to that outcome. For example, if a team working on a proposal for an innovative product does not engage in a brainstorming session and writes on the first idea that came to a member’s mind, the outcome might not be as innovative, and the learning experience not as valuable.

Since plagiarism, a common problem in writing assignments, is the means to an end (i.e. to skip the process that requires effort), one could argue that the focus on outcomes is what drives students to plagiarism. If the focus was more on the process, the learning experience might transfer better to other activities and inappropriate behaviours could be eliminated.

Most automated feedback tools, and all those that focus on summative assessment only analyze the writing product. This includes all the automated feedback system reviewed by Haswell (2006) dating back to the 1950s. It includes the four major systems of Automated Writing Evaluation software (i.e. Project Essay Grade, Intellimetric, Intelligent Essay Assessor, and E-rater) evaluated by Keith (2003) Despite being increasingly used in higher education, there is still a strong resistance to automated feedback, particularly amongst composition instructors. Haswell (2006) and others (Shermis & Burstein, 2003) have
discussed arguments both in favor and against using such tools. Many of the positive and negative claims made about how these feedback tools support learning remain largely unsubstantiated. However, a detailed discussion of this issue is not the goal of this paper. While some automated systems have been developed to be more discipline-informed, such as those designed for science and engineering students (Andeweg & De Jong, 1996; J. R. Hayes, et al., 2007), the real value of automated feedback remains contested. The techniques of Natural Language Processing (NLP) and Machine Learning used for automated writing tutors are similar to those used in Automated Essay Scoring (AES). The increasing use of these AES in many institutions has sparked debates about accuracy and pedagogical value. Two recent books discuss advances in automatic essay scoring, one taking a very supportive approach (Shermis & Burstein, 2003) and one providing a more critical perspective (Ericsson & Haswell, 2006).

Technically providing feedback on the process of writing is only possible when the document’s history is stored and mined. Collaborative writing processes (Southavilay, Yacef, & Calvo, 2010) can be studied to improve the effectiveness of a team. Fortunately many of the current writing tools (e.g. Google Docs) record the revision history in a way that we can use in feedback systems (Rafael A. Calvo, O’Rourke, Jones, Yacef, & Reimann, 2011).

2.5 Personal drivers and motivation

One reason why students do not react as expected to instructional feedback is miscommunication between student and instructors. The unexpected ways in which students understand feedback and reasons for not engaging with the feedback are also discussed by Haswell (2006). In general many authors agree that students’ reactions to feedback are very complex, affecting identity (Haswell, 2006), confidence and self-esteem (Young, 2000). These affective and motivational drivers are increasingly part of learning technologies (R. A. Calvo & D’Mello, 2011) that can be adapted during a learning activity or customized for a particular student.

It is reasonable to expect that the dimensions that learners use to assess the quality of human feedback would apply to automated feedback. Students assess the quality of feedback from peers or tutors in three core dimensions: developmental, encouragement, and fairness (Lizzio, Wilson, & Simons, 2002). The developmental dimension can be thought to include the suggestions, comments, and questions that a reader provides and that help the writer identify problems in the composition. In their study, developmental feedback showed the strongest association with quality feedback attributes (Lizzio, et al., 2002) and it is the focus
of the Glosser system described later. As a form of feedback Glosser provides trigger questions and visual representations of the document but does not try to provide an assessment based on an ideal composition.

Students conceptions of feedback, both human or computer-based, affect what they do with it and their learning outcomes (R.A. Calvo & Ellis, 2010). This evidence highlights how beyond the software’s design, its use will significantly depend on students’ prior experiences and the context of the activity.

3. Glosser Overview

Glosser was designed to provide feedback of different types, spanning the dimensions reviewed earlier: individual vs. collaborative writing, surface vs. content features, product or process feedback, multiple representations (e.g. visual or text) and delivery mechanisms (i.e. push or pull delivery) and personalizing (or not) the feedback given different individual motivational drivers.

Glosser was also designed around the principle that precise quality judgments on text features are difficult (if not impossible), so rather than being directive, the feedback should provide different perspectives (Flower, 1994) on the writing and help students engage with it.

The framework focuses on automated feedback, and the management of the actual writing activities, the peer-reviews or assessment, as well as online tutorials are covered by a separate system (Rafael A. Calvo, O’Rourke, Jones, Yacef, & Reimann, 2011). Furthermore the architecture was designed so functionalities are abstracted from their implementation and can be provided by other projects. For example, we use an open source spell checker (iSpell) that can be replaced for another without having any impact on other components.

Glosser is implemented in Java using the Spring framework that implements the Model-View-Controller (MVC) pattern frequently used for Internet applications in academic and corporate environments. The MVC pattern separates the business logic from the user interface, making it easier to modify one without affecting the other.

We first discuss the overall architecture (3.1) and its five main components (Doc, Site, Harvester, Repository, and Tool). Some tools (forms of feedback) require intensive natural language processing techniques described in 3.2.
3.1 Overall architecture

Error! Reference source not found. shows a class diagram of Glosser’s architecture. The main components are the Site, Harvester, Repository, and Tool.

![Diagram of Glosser's architecture](image)

Figure 1: Schematic of the Glosser architecture

**Doc:** A doc class is instantiated for each document in the system. The main class contains a system wide ID (and one for the indexed TML repository), and the content. It also is responsible for keeping track of the revisions, used in the tools that analyze the history of a document, and the authors and owner, used for the permission management. The Doc’s package uses a Data Access Object (DAO) to manage access to the persistent repository (the ‘database’ in Figure 1 and 2) implemented using Hibernate Object Relational Mapping. Hibernate allows different installations of Glosser to use different relational databases with only small configuration needed. The package also provides statistics produced from each document (e.g. number of paragraphs, sentences and words).

**Tool:** A ‘tool’ defines a view for analyzing a specific feature of a document. Glosser provides a simple and extensible way for new tools to be installed without any modification to the
original source code. The procedure for creating a tool requires the implementation of a Tool class, which performs an operation on the document content and returns a view. This class must be annotated with metadata, which is read by Glosser at runtime to make the tool available for use.

**Site:** Different writing activities and pedagogical approaches require different types of feedback and a Site class contains the specific configuration for each activity. This includes the list of installed tools, the language locale, any subject specific messages (for example welcome messages or trigger questions) and the document data source (e.g. Google Docs) used for writing and from which the documents need to be downloaded. An installation of Glosser can have any number of sites configured. Each Site has a Harvester thread for managing the downloading and queuing of documents and a Repository thread for managing the processing and storing of TM operations. Separating out these components allows Glosser to scale each site to a larger number of users. Moreover, each Harvester and Repository can be configured separately for each Site, this allows optimizations based on what might be known about a specific activity. For example, if the instructor managing a writing activity (i.e. Site) does not require the storage/indexing of multiple revisions (only the last one), the harvester can configured appropriately.

**Harvester:** In order to provide the most general use scenario, all the documents to which Glosser provides feedback are assumed to be on a content repository at a different system or on the cloud (e.g. Google Docs). Glosser and the document repository interact through an abstraction layer that can be used by any other component in Glosser (i.e. Harvester). In the examples and evaluations discussed here documents have been written and are maintained in Google Docs. A Harvester class encapsulates the functionality for downloading documents from a particular data source. Each Site requires a Harvester to download the document content and metadata. This design allows for additional Harvesters to be implemented, which could potentially provide support for any type of document data source or format available on the web. Currently, two Harvesters have been implemented to support documents from Google Docs and Trac’s wiki.

**Repository:** Once the document is harvested a local copy is used for processing. A Repository holds the logic for preprocessing a document’s content and storing the results of
Text Mining (TM) operations (as explained above in section 3.2. A separate Repository is configured for each activity (i.e. Site), so when the activity is finished it can be deleted freeing resources for future use. The repository provides an abstraction to indexed documents (currently using Apache’s Lucene) and a collection of text mining services. This design also allows a Repository to be customized for a Site’s language and the specific TM operations that need to be performed for its tool suite.

Since documents are downloaded from an external data source which Glosser does not control, and given the high computational intensity of the TM operations used, generating a feedback response on demand and in real-time presents a number of scalability and concurrency challenges. The evaluation section compares different architectural solutions to the scalability challenge.

When the repository keeps the revision history of a document (i.e. in Google Docs), Glosser allows users to get feedback on any particular revision of a document. This requires careful consideration when multiple courses users are attempting to download and process their documents at the same time. In order to better manage concurrency and load, the execution of document downloading and processing has been decoupled from the rest of the application. The evaluation section compares architectural solutions to this challenge.

The flow of data within Glosser is illustrated in Figure 2. It starts when a user (or a system event) triggers a document download request. First, the document is queued for downloading by a Harvester, which downloads the document content and metadata and queues it for further processing. Queued documents are then picked up by a separate process, which performs TM operations on the document content and stores the results in the Repository.
3.2 Text mining

Most of the forms of automatic feedback covered by Glosser require some type of natural language processing or machine learning to produce the features that are shown to the user. The framework architecture of Glosser aims to these Text Mining (TM) techniques (García Adeva & Calvo, 2006), both in the way they are used (i.e. designed) and the actual source code. These allows engineers building a new Tool (i.e. a form of feedback) to focus in adding what is new and reuse what is common, including harvesting of documents, local storage, indexing, parsing and machine learning (i.e. training models with existing data).

The Text Mining Library (TML) provides the reusability of indexing, parsing and machine learning aspects in Glosser. TML implements the processing of text for surface or content oriented feedback. Its designed around a five-step process:

1) Document selection, where a set of documents is selected as a corpus from a repository of documents

2) Document pre-processing, each document is decomposed in its most basic components called tokens, and some basic statistics are computed (e.g. term frequency). Some optional sub-steps can be performed at this time, like term
stemming and removing stopwords. The result of this stage is every document represented as a term frequency vector.

3) Feature extraction, each document is represented as a set of common features for the whole corpus. Features can be obtained from the term frequency vectors or extracted from the text using Information Retrieval (IR) and Computational Linguistics (CL) techniques. A common feature from IR is the combination of the frequency of words in a document and their frequency in the whole corpus, while a common feature from CL is the grammatical/syntactical structure of sentences among others.

4) Model creation (corpus analysis). The corpus is processed using data mining methods such as factorization techniques used to calculate semantic distances between documents or terms.

5) Data manipulation (operate models). The model is can be used to produce interfaces that allow the user to select other documents to process. Common tasks in this step are the clustering of documents by topics, and the prediction of categories for unclassified documents.

Each of the five processing steps can be extended with new methods and, wherever possible, reuses well tested components. For example, document selection and pre-processing (steps 1-2) use Apache Lucene to provide information retrieval methods for tokenization, stemming, stop-words removal, indexing and search. TML leverages these features by instantiating the chosen methods in configuration files that can be changed without recompiling its code. The set of documents selected for processing is called a Corpus, and it can be based on a date range, authors or keywords found in the documents.

Feature extraction (step 3) is implemented via annotators that extract information from text segments that are stored in a local repository. Each feature has a label that uniquely identifies it, and an annotator that extracts the information and associates it to its text segment source. New annotators can be added by defining a unique label and implementing the “Annotator” interface. One example is the PennTreeAnnotator class that implements the Annotator interface with the “penntree” unique label, which uses the Stanford NLP parser to extract the grammar tree from each sentence in the “Penn String” format. Model creation is an optional step that is required for content features that use more sophisticated TM algorithms.

The models created in step 5 can be based on the vector representation of some text, or on the probabilistic network of a grammar. The Corpus defines the input to build a model, for example a Corpus including the required readings for an essay task can be used to define a
Vector Space Model, which can later be used to calculate the distances between the students’ essays. New models are contributed frequently from the wider research field of Data Mining, and they can be easily tested and integrated in TML.

For evaluating models outside the production environment, TML can be used independently of Glosser and export the features extracted into Matlab and Weka formats. Another interface can be used to insert code by subclassing. For example this was used to add two implementations for matrix factorizations: Singular Value Decomposition and Non Negative Matrix Factorization (O’Rourke, et al., 2011).

The final step (5) of data manipulations can be broken into ‘operations’ on a corpus or model. These operations are not new mathematical computations but different views of the calculations made in a model. For example, calculating the semantic distance between consecutive paragraphs in an essay is implemented in TML as an operation, and the model it uses is the NMF factorization of the corpus formed by the paragraphs of an essay. The difference from this operation and its model is that instead of presenting the distances as a matrix with numbers and indexes, it does it presenting all pairs of paragraphs and distance to the previous one in consecutive order. Operations can be added to TML by implementing the Operation interface and defining a new class for its results. Many operations can be performed on a single model.

Because of the regular schedule (during a single day or the school year), and the fact that most users are in a single timezone, the workload in learning technologies such as Glosser can have a high degree of burstiness (Menasce & Almeida, 2002). Although students can submit their work at any time before the deadline; submission rates are much higher closer to the deadline.

Scalability is addressed in two ways: Decoupling the processes in the TM process, so they can be run in parallel, and using a search engine as main storage. Decoupling is shown in Figure 3, where processes are separated so the initiators change. In a standard TM process, the user initiates it by selecting a corpus, in TML there are three initiators: The user selecting a corpus, adding a document and an independent process. This change allows the three processes to run in parallel, even in different hardware platforms, facilitating scalability. TML’s main storage was also chosen for scalability, using a popular open source search engine for its implementation (Apache’s Lucene). Search engines are optimized for managing large number of documents and retrieving them based on queries.
The most basic structure in TML is a Repository, which represents all available documents and their meta-data at three levels of granularity: documents, paragraphs and sentences. The Repository class is implemented in TML using a Lucene index. When a new document is added it is split into paragraphs and then into sentences, each passage is then indexed so basic features are calculated for all of them. A set of documents in TML is called a Corpus and is created by searching a Repository using the query syntax in Lucene, for example the query 'type:document AND content:English' will find all documents containing the word 'English'.

Visualization
Prefuse and Java applets are being used.

4. Sample Feedback Tools

Figure 4 describes the key elements of a Glosser page:

- All the Tools selected by the instructor to be made available to a particular cohort in a subject.
- The Authentication details indicating the user is logged in as.

Figure 3: TML decoupled architecture
• The *Trigger questions* designed to scaffold the reflection process. Each tool comes with a set of default questions that can be customized by the instructor.

• The *revisions* that the student can access or gloss. This allows the writer to go back in the history of the document and see how a particular set of features has evolved (e.g. the topics of the composition).

• The *Gloss*, or actual feedback provided to the students. Depending on the tool this can be interactive texts or visualizations.

![Figure 4: Key components of a Glosser page.](image)

<table>
<thead>
<tr>
<th>Tool</th>
<th>Individual / Collaborative</th>
<th>Surface / Content</th>
<th>Product / Processes</th>
<th>Visual / Text</th>
<th>Push / Pull</th>
<th>Personalization? (Yes – No)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Spelling</td>
<td>I</td>
<td>S</td>
<td>P</td>
<td>T</td>
<td>P+L</td>
<td></td>
</tr>
<tr>
<td>Flow</td>
<td>I</td>
<td>C</td>
<td>P</td>
<td>T</td>
<td>P+L</td>
<td></td>
</tr>
<tr>
<td>Flow Map</td>
<td>I</td>
<td>C</td>
<td>P</td>
<td>V</td>
<td>P+L</td>
<td></td>
</tr>
<tr>
<td>Topics</td>
<td>C</td>
<td>C</td>
<td>P</td>
<td>T</td>
<td>P+L</td>
<td></td>
</tr>
<tr>
<td>Topics Map</td>
<td>C</td>
<td>C</td>
<td>P</td>
<td>V</td>
<td>P+L</td>
<td></td>
</tr>
<tr>
<td>Concept Map</td>
<td>I</td>
<td>C</td>
<td>P</td>
<td>V</td>
<td>P+L</td>
<td></td>
</tr>
</tbody>
</table>
A number of tools have been developed for Glosser and we briefly describe them here. We then provide more technical details on key tools, particularly those that address one or more of the engineering challenges mentioned earlier.

- **Spelling.** We have integrated iSpell so after a document is harvested the tool displays it with the misspelled words underlined. When clicking on the word the student can see a list of suggestions to fix the problem. This tool is a good example of the data flow from harvesting to user interaction so it is described in more detail later.

- **Flow.** Two paragraphs are said to flow when they talk about similar topics. They do not flow when the semantic distance between them is too large. Glosser’s tool displays warnings when the distance between two paragraphs is greater than a threshold (that can be changed by the user). This tool is a good example of how single document LSA can be used and is discussed in more detail later on.

- **Flow Map.** This map provides a visualization of how the parts of a document relate and follow each other, producing a visual representation of a quality measures based on the average semantic distance between parts of a document (O’Rourke, et al., 2011). The visualization is not only useful to students but has also been shown to help tutors mark essays more efficiently and reliably (O’Rourke, et al., 2011).

- **Topics.** This tool (shown in Figure 4) shows a table with the key topics developed in the document, the three most important sentences referring to that topic and the identification of the person who wrote most about it. Default trigger questions include: “Are the ideas used in the essay relevant to the question?”, “Are the ideas developed correctly?”.

- **Topic Map.** The topic map tool is used to identify the structure of topics developed in the document. The tool identifies keywords using a network-based analysis then maps them to a visual representation. If two keywords are used in the same sentence, they are linked in the visualization. The structure of topics can then be subjectively inferred from the semantics of the visualization. The sentences that cover more than one of
these topics can be seen when the link between them is shown on a rollover. The interactive visualization tool is implemented as a Java applet.

- **Concept Map.** The topic map tool uses a naïve approach, where ---- . The concept map tool incorporates an automated concept map generator discussed in more detail elsewhere (Jorge Villalon & Calvo, 2009; J. Villalon & Calvo, 2011).

- **Participation.** This tool is aimed at supporting collaborative writing activities and/or the teaching the awareness of the writing process.

- **Automatic Questions.** Questions of different types can be generated from a single document. We have developed an approach to automatic question generation (Liu, et al., 2010) integrated into this tool. Since the questions currently available are produced from citations found in the document, this tool is mostly useful for activities involving literature reviews.

4.1 Spelling (Jorge)

Surface
Low computational cost
interactivity

4.2 Concept and topic maps

Compositions represent knowledge in the form of propositions that are expressed in sentences and paragraphs. Concept Maps are cognitive visualizations (graphic representations of knowledge) made of propositions with two concepts linked by a labeled arrow. The purpose of the Concept Map Miner (CMM) is to map the former into the later.

CMM produces the maps in three steps:

1) **Identify concepts** by using syntactic information. requires essays to be split in sentences and then obtaining the part of speech for each term in the sentence, this is, identifying their role as nouns, verbs, adjectives, etc. Concepts are then identified by finding “compound nouns”, terms or phrases that include only nouns (e.g: Computer or Text Mining).

2) **Identify relationships** by using grammatical connections between the previously identified concepts. The relationship identification requires two analyses: Firstly, a grammar tree must be built from the POS tags obtained before, this is done using a probabilistic parser. A grammar tree identifies phrases within the sentence, groups of terms sharing a syntactic role. Secondly, grammatical dependencies between terms and phrases are identified using
rules. These dependencies correspond to grammatical relationships like the subject, verb and object in a sentence. The complete sequence of POS tagging, building the grammar tree and then finding dependencies has a very high computational cost.

3) **Summarize** by keeping only the most relevant concepts by ranking concepts using statistical techniques. We have reduced its computational cost by incorporating a novel technique where all terms are sorted by their rank. Ranks calculated using a semantic space produced using Latent Semantic Analysis (LSA) (T. K. Landauer, D. McNamara, S. Dennis, & W. Kintsch, 2007). LSA uses a corpus of background knowledge to build a space, which is usually very large making its calculation very expensive. Besides the cost, LSA can be be sensitive to the background knowledge used. This affects the extraction of CMs from essays because most essays are written on different topics. A novel approach to LSA is the use of Single Document Semantic Spaces (Gong & Liu, 2001), which do not use background knowledge for its calculation. This technique addresses these two issues by not using background knowledge but only the document itself. The disadvantage is that it makes the summarization unaware of anything not in the document, like disciplinary knowledge that a human expert would have. This is appropriate for situations where the reader is asked to reflect on his writing, but is different those where an expert comes to the activity to provide advice.

CMM leverages TML’s decoupled architecture for text analysis. Firstly, after documents are harvested by Glosser, they are split into sentences and paragraphs then indexed and stored in the local repository. In parallel, an annotator process runs looking for passages that need to be annotated. A particular one, the “PennTree” annotator, processes sentences and tags terms with their Part of Speech and calculates the grammar tree for the sentence storing it in the “PennString” format. Secondly, when a student clicks on the “Concept Map” tab in Glosser, a message indicating that the document is still being processed will show up in case the annotator process haven’t finished yet. If the document is ready, TML will run the CMM operation on the corpus formed by all the sentences of the document. This operation includes all three CMM subtasks which are executed to form the final CM.

Concept maps are visualized using an interactive interface that uses information in a structured XML format. Once a CM is extracted from an essay, all its information is stored as an XML structure, with a “conceptmap” element as root, a set of “concept” items, identified by a unique id and a label, and a set of “relationships” elements which have a source and target concept ids as attributes and a label for its linking word. The XML information is
translated into GraphML, a markup language used by the Prefuse Java library, which generates the maps and interactions (e.g. rearranging concepts and relationships).

The “Concept Map” tab in Glosser shows the extracted map as the gloss for students to scaffold reflection on the main concepts of their essays, and how is that they are linked. The “Concept Map” tab includes three questions:

- Do you think the most relevant concepts you covered in the essay are present in the map?
- Could you improve the Concept Map by adding or removing concepts and relationships?
- Do you think someone could understand your argument only by analyzing this map?

The focus of this tab is for the student to reflect on the whole content of her essay, focusing on the main ideas expressed in it. The first question is aimed to focus the student on reflecting on the relevance of the concepts in her essay, by identifying the most important and contrasting her ideas with the statistical results. The second question is aimed to scaffold reflection on the holistic quality of the map, and linking it to the essay by assessing if the addition of concepts not present in the map would improve its quality. Finally, the third question is aimed to focus the student to reflect on the argument of her written work, and confront it against the CM visualization. In this way the student must create an idea of her argument, which summarizes the ideas expressed in the essay and also judging the quality of the extracted CM against her idea of the argument.

### 5. Evaluation

The architecture of a software framework can be evaluated from several perspectives. From the developers’ point of view, a framework must increase reusability of design and implementation. Evaluating how well the architecture supports this requirement requires the analysis of how much more productive developers are while using it. So far our framework has been used to produce eight tools, by a small number of developers (six), so the numbers are not significant enough for an objective evaluation.

Another key requirement for a framework is that its architectural design and its implementation is shown to be able to scale to the number of expected users. This analysis requires workload characterization, modeling and a framework for capacity planning, all discussed in Section 5.1.

In addition to the above evaluation parameters, Glosser as a learning tool should be assessed on the impact it has on learners and instructors. Section 5.2 discusses students
perceptions of Glosser, comparing it to their perceptions of human feedback. Analyzing students’ experiences of learning provides an e-learning software development approach [REF]. E-learning tools also have an impact on how instructors teach and assess, for example it can make tasks such as assessing an essay more or less efficient and reliable. Section 5.3 discusses a study on how one of Glosser’s visualization tools affects instructors’ assessment.

5.2 Workload characterization and modeling

The first step towards a workload description. In the present context this refers to how students interact with Glosser (through the iWrite system). iWrite has a number of students (#students), enrolled in subjects that use it for writing activities, as a reference (through the online tutorials it contains) or both. These students are a subset of the total in the university or the Faculty. The target audience for iWrite was the Faculty of Engineering and IT that has 3,000 students, and iWrite has about 500 per semester. The number of students depends on the commitment of instructors to explicitly teaching academic writing. Each students may engage in one or more activities using iWrite (#assignments) and some of them would have Glosser as a support tool (#GlosserRatio), others may have peer-feedback, tutor-feedback or none.

In this paper we are interested in the functional workload description of Glosser, and how it impacts capacity planning. Using the definitions above the amount of GlosserTransactions is:

\[ \text{GlosserTransactions}(t) = f(\text{students} \times \text{GlosserRatio} \times \text{assignments},t) \]

There are two important features for f. Web transaction logs are generally bursty, with high peaks of workload that are much higher than the average. Second, different functionalities will have different usage patterns and different resource requirements. These two features can be evaluated with real transaction data.

A model of the workload was produced by analyzing log data collected during semester 2, 2010. We had 500 students and the GlosserRatio was 0.1. Students had 2 assignments where Glosser was available. The number of assignments, particularly deadlines, is important to the analysis because usage of the tool often peaks close to the deadlines. These peaks produce a burstiness effect [REF], where at certain times, the amount of traffic can be hundreds of times greater than the average. Figure 5 shows how the number of page views for a group of students using Glosser on 1-minute epochs. The maximum number of transactions maxBurst = 17 while the minimum was 0 and the average avgBurst =
0.0082. The burstiness coefficient \(a\text{Ratio} = 2073\) (max/avg). The length of the epochs affects the results. For this same data when the epoch is set to 1 hour, the \(\text{maxBurst} = 191\) and the \(a\text{Ratio} = 388\). Burstiness effects at different time scales has been shown in earlier research, what is significant here is that the ratios can be quite high and have a strong impact on the performance of the system.

Out of these students, only 50 were asked by their instructors to use Glosser. Figure 6 describes the navigation patterns that these users followed. The home page is, as expected, the most visited. Only a third went beyond this page that contains basic statistical information. The topics map was the most used functionality, followed by flow and participation.

The third level of workload analysis is a resource-oriented description. Glosser basic components (i.e. classes where workload is homogeneous) are the Harvester, TML, the Tools (web application) and visualizations (within the tools), each with different resource usage patterns. Since we did not record resource usage over the period that the above data was collected, the burstiness and usage distribution above were used to simulate loads.

**Resource Usage**: the maximum CPU and I/O time of request can be labeled as Trivial, Light, Medium and Heavy. If harvesting is done *apriori* I/O time is minimal, as the size of the files returned does not exceed an amount.

**Internet applications**: Glosser only uses HTTP. iWrite running on the same server sends emails to users using SMTP, but the load of this operation is minimal.

**Geographical orientation**: Since all users had to be students enrolled in a subject at the University of Sydney, the vast majority used the system from the same location as the server.
Figure 5: number of concurrent Glosser sessions in one class.
The workload testing was based on the basis of varying #students (to 10,000, well beyond the total for our faculty). Given the burstiness and use patterns observed we estimated that 50% of the students could try to access it at the same time in a course, but the different courses would have deadlines on different days.

\[
#\text{students} \times \text{GlosserRatio} \times \text{peak}_\text{use} = 10,000 \times 0.5 \times 0.1 = 500
\]

This experiment was designed to evaluate the effectiveness of the Glosser architecture in delivering on-demand automatic feedback. The architecture was evaluated under the load of 200 users simultaneously requesting documents on two separate sites (100 users per site). The content of the documents are identical with each being 545 words in length (roughly the length a short essay).

Site 1 was configured to index documents without any TML annotators. Site 2 was configured with the computationally intensive Penn tree annotator, which calculates and the linguistic structure of a document and annotates it with part-of-speech tags. The time take to harvest (download) and index each document was recorded from the time of the initial document request.

Figure 1 charts the time taken for each site to harvest and index 100 documents. The chart shows that the time to harvest and index a document increases linearly per site as the number of document requests increases. Each site took approximately 2 seconds on average to harvest a document from Google Docs. Site 1 took 0.28 seconds on average to index a document. Site 2 took 10.84 seconds on average to index a document. The different in indexing time between the two sites can be attributed to the additional processing required by Penn tree annotator configured in Site 2.

The results show that the Glosser architecture can scale to provide feedback using computationally intensive text mining algorithms to multiple courses of students within a usable time period. The architecture was able to balance the distribution of computational resources among the sites, while not locking up resources or overloading the server.

The ability of Glosser to effectively scale is due to the design choice of queuing documents for harvesting/indexing. This ensures that Glosser does not become overloaded with document requests in times of high load (such as close to an assignment deadline). Furthermore, having documents queued on a per site basis (rather than a per user) ensures that
sites configured to preprocess and store the results of computationally intensive text mining operations do not block users of other sites from harvesting/indexing documents in the meantime. This also ensures a fair distribution of hardware resources among each site. Finally, decoupling the harvester and indexer of each site in separate threads can also reduce the wait time between downloading and indexing in times of high load.

![Graph showing time take for each site to concurrently process 100 document requests.](image)

**Figure 1: Time take for each site to concurrently process 100 document requests.**

### 5.2 Students’ conceptions of automated feedback (and why they matter)

Students’ conceptions of feedback from tutors and automated systems like Glosser are related to achievement. We investigated (R.A. Calvo & Ellis, 2010) these relationships within the context of teaching academic writing in an Engineering subject. The study had students (N=46) who worked in pairs to write an engineering report on e-business. The design of the study involves in-depth interviews (N=22), and the analysis employs an approach in which student conceptions of automated feedback are investigated in relation to related feedback from their tutor, perceptions of automated feedback in general and their academic achievement. The study showed that Students’ conceptions of feedback vary and can be grouped into cohesive and fragmented, in line with other theoretical models. Close
associations were found between more cohesive conceptions of feedback and better academic performance. A student’s conception of traditional and automated feedback is similar, being either cohesive or fragmented. Changing one may change the other. Deep learners see feedback as way of learning about the topic, whereas shallow learners see them as a way to improve the communication aspects of writing. Design considerations based on these results are discussed.

These study offers a description of the way students report thinking about feedback in writing. These results to inform new technology and pedagogical design strategies for the provision of feedback that take into account the importance and impact of student conceptions.

6. References


