A Domain-Specific Approach to the Development of Ontology-Based Document Assessment Assistants

Arturo J. Sánchez-Ruíz
School of Computing – University of North Florida
1 UNF Drive, Jacksonville, FL 32246, USA
+1-904-620-1314
asanchez@unf.edu

Bart Welling
English Department – University of North Florida
1 UNF Drive, Jacksonville, FL 32246, USA
+1-904-620-1268
bhwellin@unf.edu

ABSTRACT
The second author of this paper incrementally developed, over the years, a manual process to systematically evaluate English essays by maintaining an ontology of comments crafted with the goal of pinpointing the occurrence of mined negative writing patterns (i.e., those whose use is discouraged), as well as positive ones (i.e., those whose use is praised). In this paper we report on how the automation of this process, using domain-specific approaches, led to the development of an ontology-based assessment assistant for a specific word processing system. We also report on our approach, currently underway, to extending this solution to a product line of ontology-based assessment assistants over a family of word processing systems and course management systems.

Categories and Subject Descriptors
D.2.6 [**Software Engineering**]: Programming Environments – *integrated environments, interactive environments.* D.2.11 [**Software Engineering**]: Software Architectures – *domain-specific architectures, patterns.* D.2.13 [**Software Engineering**]: Reusable Software – *domain engineering, patterns.*

General Terms
Management, Design, Human Factors.

Keywords
domain-specific software development; document assessment; software assistants; ontology-based software; software product lines.

1. INTRODUCTION
The inception of this project can be traced back to an informal conversation between the two authors in the summer of 2006, during which the second author related an assessment method he had developed over the years after having evaluated hundreds of English essays. In Section 2, the second author explains how he synthesized his method.

The first author saw the possibility of completely automating this method by constructing a software assistant which would enable evaluators to systematically, consistently, and efficiently assess documents from various domains, not just English essays, e.g., legal, governmental, medical, technical, et cetera. For instance, the assistant would enable evaluators to maintain and refine their knowledge of observed writing patterns used by authors on their documents, positive and negative alike, a practice which promotes uniformity in assessment, and the sharing of such knowledge.

This paper reports on three evolution stages associated with this project. In its first stage the project was a capstone-like course assignment. In its second stage, the project was a product developed for a specific word processing system. Finally, in its third stage, the project—currently underway—is to develop a product line of ontology-based assessment assistants, over a family of word processing products and course management systems typically used in educational institutions, which give end-users the feeling of working with a unified tool which assists them in the management, delivery, and assessment of documents with minimal context switching.

Section 3 shows the domain analysis associated with the first stage of this project which allowed us to identify the major components of the system and suggested strategies to incrementally develop the project.

Section 4 discusses the software architecture associated with the third stage of this project.

Section 5 discusses implementation details associated with the development of an assessment assistant which targets the Microsoft Word product.

Section 6 presents our approach to generalize this specific solution to a software product line over various word processing products and course management systems.

Section 7 compares our approach with others with respect to: the original goals of the assistant as envisioned by the second author; course management systems; and approaches to build software product lines.

The paper ends with our conclusions and with references to the literature we consulted.

2. User's Story
Shortly after being hired to teach literature at the university level, the second author realized that much of his time outside the classroom would be spent grading student essays. Unfortunately, too many of these essays exhibited exactly the same negative writing patterns: weak and poorly structured arguments; ineffective handling of primary evidence; incorrect citation of outside sources; unclear, clichéd, or overly simplistic sentence structures and word choices; and major, copious errors in spelling, grammar, and punctuation. Commenting on these problems in the traditional way, with pen in hand, proved to be not only labor-intensive and frustrating but pedagogically inefficient as well.

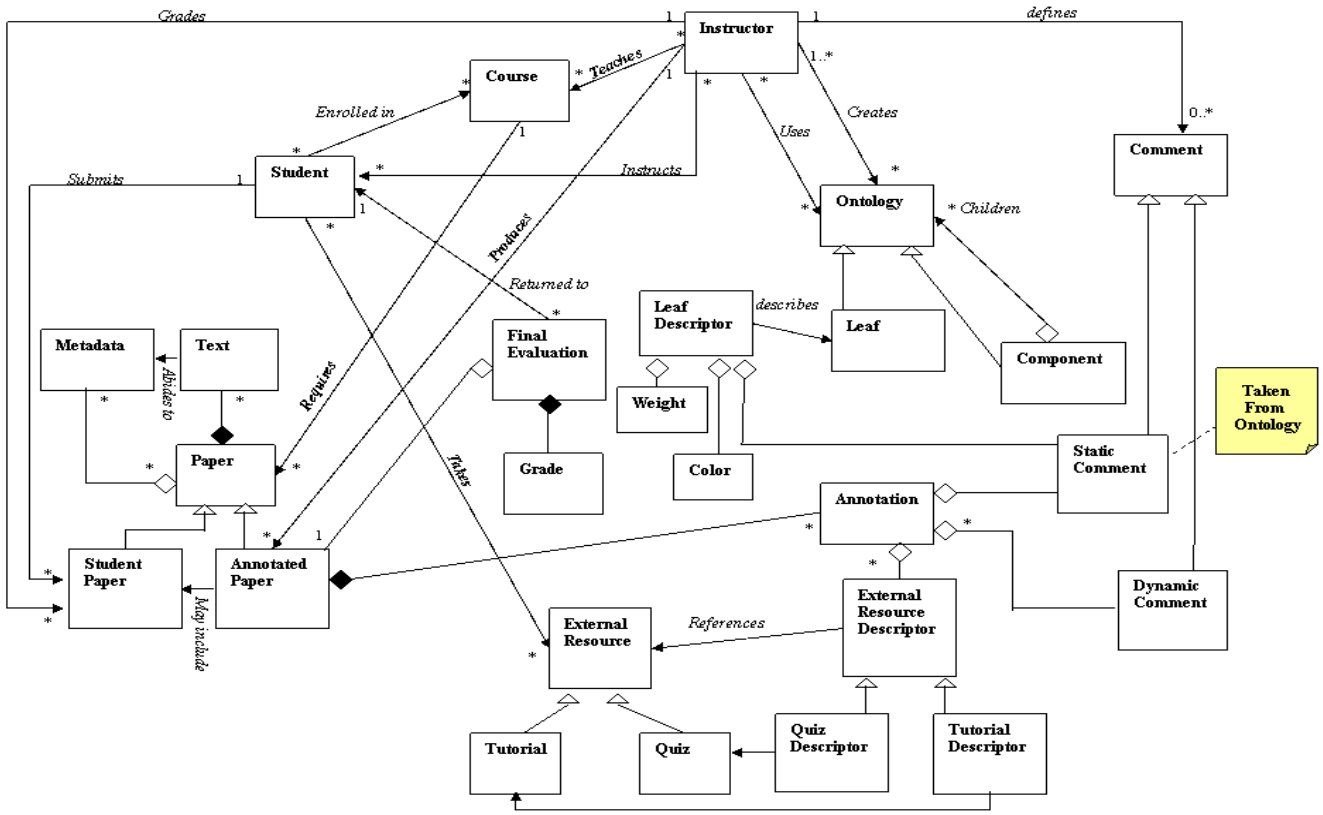

Figure 1. Domain model as an UML class diagram.

Students with the most serious writing challenges seemed to feel overwhelmed by the sheer number of comments, and given the space constraints of handwritten comments it was frequently possible to highlight only what was wrong with a given feature of a student's essay rather than how to fix it—or, better yet, how to avoid the problem in the future.

Students also complained about what they perceived as mixed messages from teachers in different academic departments, and sometimes within the same department, regarding the university's criteria for strong student writing. It quickly became apparent to the second author that students and faculty members alike lacked a common critical vocabulary for evaluating, and improving, the quality of written communication at the university. Moreover, they lacked a method to help formalize, standardize, streamline, and enrich the process of communicating with each other about student writing skills. Microsoft Word's Comment feature presented some advantages over handwritten comments, but in the end it still required the user to type the same comments in paper after paper, and time constraints made it difficult to add comments of more than a few words in length. More seriously, Word's Comment feature was not designed to link comments together within a pedagogically meaningful and effective framework of ideas regarding the nature of strong, clear writing.

The second author began developing a set of "Grading Codes"[1] that could be inserted directly in students' essays either by hand or via a Microsoft Word comment. The codes consist of short, simple combinations of letters and numbers which are keyed to detailed comments on virtually every aspect of student writing. For instance, the code "CL-1-b" corresponds to the comment "While at one point this sentence would have been considered debatable (i.e., a claim), it would now be accepted as true (i.e., a fact) by the majority of scholars, and should thus be rendered more debatable." The codes comprise an ontology that can be expanded every time the second author encounters a previously unknown error in his students' essays. Indeed, like the Linnaean taxonomic hierarchy, the "Grading Codes" have the potential to be infinitely expandable. However, the greatest strength of the system is also its greatest weakness. As the list of "Grading Codes" grows, it takes more and more time for students to decipher the comments on their papers, and the risk that they will simply ignore the codes increases.

After discussing the preliminary version of the "Grading Codes" with the first author, both authors realized that an automated assessment assistant combining the best features the "Grading Codes," and Microsoft Word's Comment feature would present the ideal solution to these problems. A software

[1] A sample of these codes can be downloaded from http://www.unf.edu/~asanchez/dsm08

application that interoperated with Microsoft Word would enable the user to add detailed pre-written comments to documents quickly and exactly where they were needed. Precise and transparent point values (determined and revised, as needed, by evaluators) could be assigned to different positive and negative writing patterns. The application, like the "Grading Codes," could be based on an infinitely expandable and flexible ontology, but students—while familiar with the ontology, and thus with the standards against which their writing was being evaluated—would not be distracted by the need to engage in context switching between a set of codes and the comments they stood for.

The assessment assistant could also incentivize student learning by means of hyperlinks to online tutorials and quizzes, the successful completion of which could translate into a higher grade on the essay. Not only would such an application foster clearer communication between teachers and students vis-à-vis student writing, but it could benefit academic departments and entire universities by facilitating greater uniformity in grading practices, by providing a mechanism for sharing customized grading ontologies, and by enabling academic units to track and respond more effectively to emerging trends in student writing. Finally, since users would be able to revise ontologies freely, the application could potentially streamline and enhance the document assessment process in any number of fields beyond the walls of academia.

3. DOMAIN ANALYSIS

The result of our initial domain analysis, using the approach discussed in the book by Larman [5], is presented in Figure 1 as an UML diagram [6]. Arrows with a small head represent general association relationships (labeled); arrows with a larger head represent "is-a" relationships (i.e., generalization/specialization); clear diamonds represent aggregation relationships; dark diamonds represent composition relationships; and boxes represent relevant domain concepts. Some multiplicities are shown.

This initial analysis allowed us to identify two major sub-systems: document management and delivery; and document assessment. The former interfaces with course management systems. The latter interfaces with word processing systems and maintains ontologies created by evaluators. In the next section we show these components depicted as an architectural diagram.

4. SOFTWARE ARCHITECTURE

Figure 2 presents a view of how the major subsystems identified by our domain analysis interact, from the perspective of end-users: document authors and document evaluators. The acronym ISA stands for Integrated Software Assistant.

We refer to the encircled numbers in Figure 2 to illustrate a typical use case. Authors prepare their documents using some word processing system, which we abbreviate as WPS (1). Authors submit their documents to the evaluators via the course management system, which we abbreviate as CMS (2). When evaluators launch the ISA on their computers, it connects to the CMS and determines whether documents are ready to be downloaded to their computers (e.g., if the current date is past the deadline associated with this coursework). In this case, documents are downloaded, typically as a single archive—e.g., zip, rar, flex, etc.—uncompressed and stored in a well-defined place from which ISA can retrieve them. At this point ISA is also keeping track of who submitted what, e.g., in connection with a coursework. The main user interface metaphor associated with ISA is that of a "dashboard" which contains icons suggesting various tasks evaluators can perform (3).

When evaluators are ready to start assessing the documents, they interact with ISA through the dashboard and additional dialog boxes, shown as needed. For a single document, the flow of activities is as follows. ISA opens up the document to be evaluated in the WPS. As evaluators identify the occurrence of a writing pattern, they select the text, and then the corresponding comment from the ontology. Evaluators can use multiple ontologies and update them in the middle of the evaluation process at will (4).

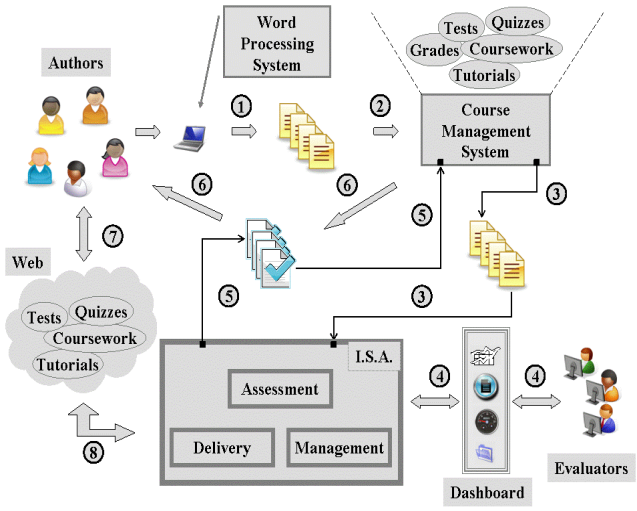

Figure 2. Software architecture: end-users' view.

When evaluators let ISA know they have finished evaluating all documents, ISA packs them as a whole, and sends them to the CMS. The final effect of this interoperation between ISA and the CMS is that documents are stored in pre-defined places in the CMS from which authors can retrieve them, and document grades (e.g., in the case of a course) are recorded as per the CMS's conventions, which become available for students to browse (5).

Authors connect to the CMS and retrieve their documents. To them the retrieved WPS document is just the original submitted document augmented with comments—inserted by ISA as per the evaluator's actions—and an extra page at the end with a summary of the evaluation, which includes the grade associated with this document, when applicable (6).

Since comments in the evaluated documents can contain embedded hyperlinks, ISA knows (for instance): if an author visited a website with a learning instrument (e.g., tutorial, quiz, and test); the author's identity; if the author completed the instrument; and a summary of the author's attempt (7). This is recorded by ISA as an assessment activity, which is transferred to

the CMS, and is therefore part of the author's assessment record, when applicable (8).

Various important design decisions were derived from the analysis of this software architecture:

[DD.1] The graphical user interface (GUI) exists in its own layer.

[DD.2] ISA interacts with the CMS via an interoperation layer.

[DD.3] ISA interacts with the WPS via an interoperation layer. This implies the solution should not be implemented as an add-in or plug-in to the WPS.

[DD.4] Ontology maintenance is independent of GUI, CMS, and WPS.

5. THE ONTOLOGY-BASED ASSESSMENT ASSISTANT FOR MS WORD

The first developed product is a specialization of the architecture in Figure 2 such that: WPS is Microsoft's Word (MSW), and ISA is just the Assessment component. The next two sections discuss implementation details associated with the Ontology Manager and the Interoperation Layer with MSW. The last section shows some screen shots of the product we developed.

5.1 Ontology Manager

Writing pattern ontologies are implemented as taxonomies. A taxonomy has a root which names the whole artifact. Under the root there are nodes which are either internal or terminal. Nodes are related by the category-subcategory relationship. Internal nodes must have descendants, which can be either internal or terminal nodes. Terminal nodes do not have descendants. The information associated with internal nodes is: the name of the category and references to descendants. The information associated with terminal node includes, but is not limited to: the name of the final category, the comment associated with the pattern, its weight, and URL's to external instructional resources.

From the perspective of the user, taxonomies can be directly manipulated via operations on categories which include: insert, delete, edit, and move. They can also be imported and exported. Taxonomies can be in two states: design, and publish. The first mode characterizes a work in progress. The second mode characterizes a finished product. Taxonomies can be transitioned from one mode to the other. They are presented to the user as a hierarchical structure with nodes that can be expanded, contracted, and moved.

Internally, taxonomies are represented as XML files with an associated schema. Imported taxonomies are checked against such a schema. Finally, schema well-formedness is the criterion used to determine whether taxonomies can be transitioned from design to publish mode.

5.2 Interoperation Layer with MS Word

Visual Studio Tools for Office (VSTO), currently in its 2005 version[2], enhances the popular MS Integrated Development Environment (IDE), Visual Studio[3], by enabling the seamless run-time interoperation between MS Office[4] applications and solutions built with the IDE.

The so-called Primary Interop Assemblies (PIA) act as the interoperation layer between our Assessment Assistant and the MS Word application. The PIA exposes the MS Word application itself and its run-time object model through .NET managed code. We decided (c.f. Design Decision [DD.3] on Section 4) to implement the Assessment Assistant as a separate application from MS Word, which interoperates with it via the PIA. The book by Carter and Lippert, as well as the book by Bruney discuss other viable programming models [1, 2]. The existence of the PIA for MS Word implied we did not need to implement the interoperation layer ourselves.

5.3 Screen Shots of our Assessment Assistant

We built a self-contained installer that checks whether the host computer has the correct versions of MS Word (2003 Professional or newer), .NET framework (version 1.1 and newer), and the PIA which correspond to these two components. The installer sets up specific folders where design/publish ontologies are kept.

Figure 3 shows the first interaction dialog the user sees when the application is launched. Figure 4 shows the options made available to the user after s/he has chosen "Manage Taxonomy".

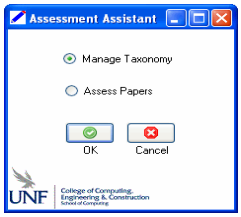

Figure 3. Initial options.

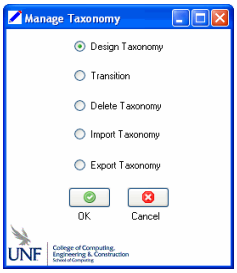

Figure 4. Manage Taxonomy options.

Figure 5 shows what the user sees after s/he has selected to continue working on an existing ontology in design mode, and has chosen the desired one. Notice the nodes can be expanded and contracted. Also notice the various options available.

[2] Visit the VSTO Developer Portal at http://msdn.microsoft.com/en-us/office/aa905533.aspx

[3] Visit the Visual Studio Developer Center at http://msdn.microsoft.com/en-us/vstudio/default.aspx

[4] Visit the MS Developer Center at http://msdn.microsoft.com/en-us/office/default.aspx?PHPSESSID=388057524368e3818e5a18783b5bd3fc

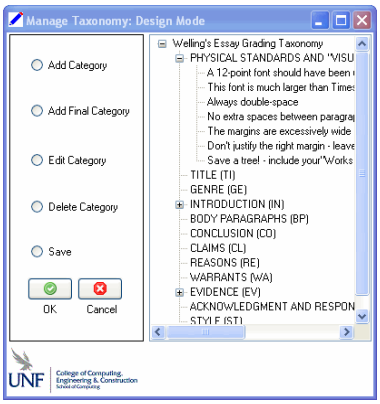

Figure 5. Working with an ontology in design mode.

When the user is ready to start assessing a document, s/he must first select the ontologies that will be used. After that, if the user chooses the "Assess Paper" option, then s/he can open as many documents as needed, and the ontologies dialog box remains open (see Figure 6, which shows two open ontologies).

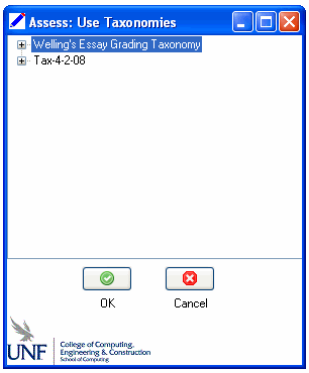

Figure 6. Ontologies to be used by the evaluator.

The user can switch back and forth among open documents, and close them at will. Only one document has the focus at any moment. Suppose the evaluator is now assessing the document with the focus, and s/he identifies a pattern. The user then selects the portion of the text with the mouse (left-click-hold-and-drag), goes to the ontologies dialog box to locate the appropriate pattern, clicks on it, and then the assistant automatically inserts a comment which contains all the information associated with the chosen terminal node in the ontology (see Figure 7).

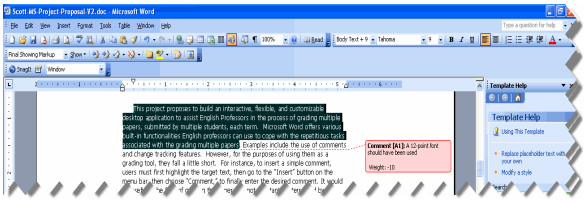

Figure 7. Pattern from the ontology inserted by the assistant.

When the user decides s/he has finished assessing the paper, the assistant generates a summary page and appends it to the document (See Figure 8). The user can revisit any evaluated paper at any moment if, for instance, s/he decides to change some of the comments and/or the associated weights.

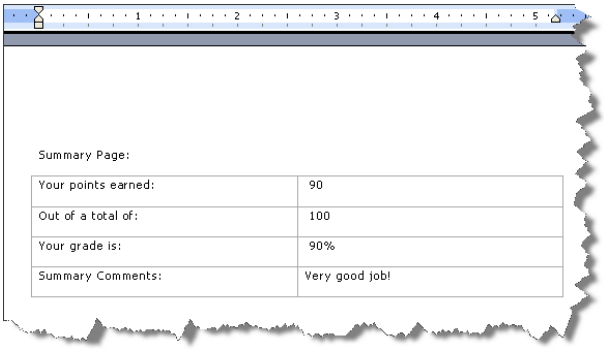

Figure 8. Summary page generated by the assistant.

6. TOWARDS A SOFTWARE PRODUCT LINE OF ONTOLOGY-BASED ASSISTANTS

A software product line is defined by Clements and Northrop as "a set of software-intensive systems sharing a common, managed set of features that satisfy the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way" [3].

From the perspective of the user, the common core of our product line is constituted, on the one hand, by a consistent user interface, hiding implementation details associated with desired features; and on the other hand, by the management of ontologies used in the assessment process. This common core is implemented, fundamentally, through a language of gestures—e.g., highlighting, pointing-and-clicking, clicking-and-dragging, et cetera—which allows users to directly manipulate ontologies and documents.

The variability of this product line can be projected onto two orthogonal axes, namely: that which characterizes the kind of word processor used by authors to compose their documents, and that which characterizes the course management system that frames the whole compose-assess-return document lifecycle.

We are therefore interested in assembling a family of ontology-based document assessment assistants as instances of the architecture discussed in Section 4. The targeted word processing systems (WPS) are OpenOffice and Acrobat. The targeted course management systems (CMS) are Blackboard[5], and Moodle[6].

To build the WPS interoperation layer (c.f. Section 4), we follow a reverse engineering approach. Namely, we first extract the calls to the Microsoft Office PIA (c.f. Section 5.2) and then build implementations of these calls against the Application

[5] Visit the Blackboard Developer Center at http://www.blackboard.com/extend/dev/

[6] Visit the Moodle portal at http://moodle.org/

Programming Interfaces (API) for OpenOffice[7] and Acrobat[8]. To build the CMS interoperation layer we follow a forward approach, namely we first define the appropriate interfaces and then implement them against the API's for the corresponding systems. This is because the operations associated with this layer are easy to understand as extensions of input/output services.

7. RELATED WORK

From the perspective of the Assessment Assistant's original goals as envisioned by the second author, we compared our approach with products from different educational technology companies to compile the information presented in a table (not shown here). We spoke directly with representatives from Vantage Learning (product: IntelliMetric "intelligent" Automated Essay Scoring System), Pearson Knowledge Technologies (product: Knowledge Analysis Technologies—KAT—Engine), and Idea Works (product: SAGrader). We also contacted Educational Testing Services (products: Criterion, E-Rater, Critique, and C-Rater), but they did not respond to our requests for more information.

The rubric used to compose the table is the following: (a) the tool automatically evaluates a document based on such criteria as strength of argument, structure, style, grammar, and/or spelling; (b) comments are added directly to the document; (c) evaluator's comments are distinguished from writer's text using a word processing system's comment tool (when available); (d) the tool fully interoperates with a word processing system; (e) the tool is potentially or currently applicable in a wide range of academic disciplines and business environments; (f) individual comments can be customized by the evaluator without outside help; (g) comments can be inserted quickly in documents; (h) comments are part of a larger ontology; (i) users can create a new ontology without outside help; (j) users can automatically import an ontology created by other users; (k) preexisting ontologies can be customized by the user; (l) an ontology can be exported to other users; (m) the tool employs an assessment metaphor that shows users how many texts have been evaluated, and in general it shows the progress of the evaluation process for a large set of documents; (n) the tool assigns point values to different patterns in the document; (o) the tool provides writers with an assessment page and/or grade; (p) the tool provides evaluators with a statistical analysis of assessed texts; (q) the tool implements its own unique Graphical User Interface (GUI); (r) the tool accommodates multiple languages; (s) the tool attempts to eliminate human-introduced errors, biases, and inconsistencies; and (t) the tool flags "problem essays," i.e., essays that cannot be scored by computer.

Our Assessment Assistant is fundamentally different from the programs produced by Vantage Learning, Pearson Knowledge Technologies, Idea Works, and Educational Testing Services in that it (1) does not attempt to evaluate the content of texts automatically; (2) uses word processing system's comments feature to directly add customizable comments to texts produced in said word processing system; (3) allows users to create, import, export, and customize ontologies without outside help; (4) employs an assessment metaphor that shows users how many texts have been evaluated; (5) does not attempt to eliminate human-introduced errors, biases, and inconsistencies; and (6) does not attempt to flag "problem essays," i.e., essays that cannot be scored by computer.

Notice that criteria (a), (s), and (t) require tools to deal with the semantics of submitted text, which our tool does not attempt to deal with. Our Assessment Assistant is a true "assistive tool" in the sense that it enables evaluators to perform their task faster in a consistent and systematic way. Our tool does not attempt to usurp evaluators and their expertise in the identification of used patterns and anti-patterns, an approach which has been the target of heated debates among members of the educational community.

From the perspective of features provided to end-users by course management systems, Blackboard (Bb) and Moodle do offer functionality that supports the management and delivery of document-centric coursework. However, the use of this functionality imposes an artificial context-switching on its users. It also potentially imposes the need to manually perform mechanical chores such as connecting to Bb/Moodle in order to download coursework, disconnect, do the preparation needed to start grading, grade, and finally connect again to upload the papers and also to upload the grades. With ISA, the user only needs to deal with one friendly and intuitive application for the assessment, management and delivery of the coursework in cooperation with the same tools students used to create and deliver it. The whole roundtrip, which starts with the submission of a paper and ends with a student receiving the graded paper and potentially taking advantage of additional learning instruments associated with the submitted paper, can be done with virtually no context-switching.

Bb also supports the development of assessment instruments such as tests and quizzes with the corresponding tracking of attempts made by students, by itself or by cooperating with other tools such as "Respondus"[9]. However, it currently does not support the tracking of attempts initiated from outside Bb. In this case ISA acts as a middleware between the users (teachers and students) and Bb as a provider of assessment instruments, therefore minimizing the context-switching for these users. This means users will continue using the features Bb provides and ISA will act as the mediator that helps users take advantage of these resources with minimal context-switching. In conclusion, the intention of this project is not to supplant existing WPS or CMS, but to build a software assistant that cooperates with them to offer end-users an extended, integrated, more powerful, friendlier, more efficient, and more effective tool.

We identify our approach to building a software product line of ontology-based assessment assistants, currently under development, with the "Product Parts" and "Assembly Lines" patterns documented in [3]. The first pattern is used in the building of core assets and the second is used when assembling core assets to produce an Assessment Assistant for specific variability choices. We also identify common points with the "use case development strategies" labeled as "forward evolutionary

[7] Visit the OpenOffice Developer's Wiki at http://wiki.services.openoffice.org/wiki/Main_Page#Getting_started_with_OOo_development

[8] Visit the Adobe Developer Connection at http://www.adobe.com/devnet/acrobat/

[9] See http://www.respondus.com/

engineering" and "reverse evolutionary engineering" by Gomaa [4], as described in Section 6.

8. CONCLUSIONS

This paper has presented an evolutionary approach to the building of a product line of ontology-based assessment software assistants. The first stage of the evolution used a domain-specific approach to identify the suite of concepts, their relationships, and operations with which end-users are familiar: documents, assessment ontologies, and an assessment process which directly manipulates documents and ontologies through gestures such as highlighting, pointing-and-clicking, clicking-and-dragging, et cetera. At this stage we designed a software architecture with clearly separated concerns: user interface, ontology management, document assessment, and document management.

The second stage of the evolution implemented an instance of this architecture by focusing on word-processing-system-neutral components, i.e., user interface and ontology management, and a word processing system interoperation layer specifically aimed at MS Word.

The third stage of the evolution, currently underway, implements the software architecture by taking into account the variability axes introduced by classes of word processing systems, and classes of course management system; via a combination of reverse and forward engineering. The target result is a family of ontology-based assessment assistants.

With respect to our application of domain-specific techniques to the development of this project, we would like to highlight the following. First, since our end-users—evaluators—directly manipulate objects naturally occurring in their domain of application—documents and ontologies—through a language of gestures, the elicited Domain-Specific Language (DSL) has therefore a non-textual syntax, which we did not formally define simply because we did not consider it a crucial contribution to the development of the project.

Second, this DSL is supported by two meta-models: the ontology meta-model (OMM) and the document meta-model (DMM). The OMM has been defined in Section 5.1 as a family of taxonomies with the category-subcategory relationship, and also illustrated as an instance of the Composite Design Pattern in Figure 1 (see the portion which contains "Ontology", "Leaf", and "Component"). Interestingly enough, the DMM is the MS Word Object Model [2].

Third—and final, the semantics of these models, i.e., what give meaning to a gesture such as "left-click-hold-and-drag" on a portion of a document—for instance, are given by the Primary Interop Assemblies in the case of DMM, and our implementation of the ontology manager in the case of OMM.

With respect to the level of generality of our approach, from the perspective of the domain of expertise associated with the documents to be assessed, our main argument is this: since the ontologies are created by domain experts—e.g., lawyers, accountants, physicians, et cetera—it is incumbent upon these experts, not upon our tool, to make sure the ontologies capture the appropriate writing patterns and anti-patterns. Experts are assisted by the tool, not substituted by it. This is why, in our opinion, the current state of the art of the area referred to as "Ontology Learning from Text"[10] is not applicable to the problem of automatically mining writing pattern ontologies from actual documents produced by authors. However, we do agree such an approach is worth exploring.

ACKNOWLEDGEMENTS

The development of this project was the subject of study in the sequence of graduate courses Engineering of Software I and II, offered by the School of Computing, University of North Florida, in the fall semester of 2006 and spring semester of 2007, respectively. We thank the students of this course for their contribution to the project (in alphabetical order): Lucas Downard, Swapna Mekala, David Scott, Sweta Shah, and Smitha Thomas. David Scott implemented the current version of the Assessment Assistant for Word as part of his MS project. The University of North Florida's Board of Trustees has registered a copyright in connection with this project.

The authors would also like to thank the anonymous reviewers of this paper for their valuable comments.

9. REFERENCES

[1] Bruney, A. 2006. Professional VSTO 2005: Visual Studio 2005 Tools for Office (Programmer to Programmer). Wrox.

[2] Carter, E., Lippert, E. 2006. Visual Studio Tools for Office: Using C# with Excel, Word, Outlook, and InfoPath (First Edition). Addison-Wesley. Note: A second edition of this book will be published in 2009.

[3] Clements, P., Northrop, L. 2002. Software Product Lines: Practices and Patterns. SEI Series in Software Engineering. Addison-Wesley.

[4] Gomaa, H. 2004. Designing Software Product Lines with UML: From Use Cases to Pattern-Based Software Architectures. Addison-Wesley.

[5] Larman, C. 2004. Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development (3rd Edition). Prentice-Hall.

[6] Rumbaugh, J., Jacobson, I. and Booch, G. 2005. The Unified Modeling Language User Guide (2nd Edition). Addison-Wesley Professional.

[10] We are referring here to the books "Ontology Learning and Population from Text", by Philipp Cimiano (Springer); and "Ontology Learning from Text: Methods, Evaluations, and Applications", edited by Paul Buitelaar, Philipp Cimiano, and Bernardo Magnini (IOS Press). We look forward to reading the proceedings of the 3rd Workshop on Ontology Learning and Population, held in July of this year.