From Documents to Objects

An Overview of LiveDoc

James R. Miller and Thomas Bonura

One of the changes that the World Wide Web has brought to the computing industry is a new way of thinking about documents. Traditionally, documents have been seen as simple streams of characters, like those in a document editor. Applications that manage these documents may do more or less interesting things to the characters, but they rarely attempt to interpret any of the meaning of the document. There's obviously meaning there, but it only becomes apparent when read or otherwise manipulated by a human. In contrast, the Web has brought with it the concept of a document that has been authored in such a way that important bits of information are explicitly identified within the document. This identification exposes some of the meaning of the document, albeit at a fairly low level, so that various kinds of actions – primarily "show me this related document" – are offered to users and made easy for them to carry out.

The gap that separates these two notions of *document* is the need for the human authoring of the Web document. More to the point, it's the need for a human to identify the meaningful components of the document and the actions that make sense for those components. There is a real opportunity to advance the computing field here, by bringing these two worlds together: by enabling an ordinary document, built with any application, to automatically offer users access to some of the meaningful bits of its content, and by helping users carry out appropriate actions on these objects.

Bridging the Gap through Structure Detection

This premise led to a collection of projects within Apple's Advanced Technology Group – within the Intelligent Systems Program, in particular – on the idea of *structure detection*. The work was based on the observation that, while automatically computing a high-level understanding of an arbitrary document is beyond our present ability, many meaningful bits of information are computationally quite easy to recognize: recognizing an e-mail address ("fred@apple.com") or a URL ("http://www.apple.com") takes little more than a context-free grammar, if not merely a regular expression parser. A first step to bridging the document gap described above is then to construct a means of passing text from a user's document into a parser for matching against a collection of recognizers, each of which is looking for some meaningful type of information. These identifications imply simple interpretations of the bits of information that were found: URLs are found by the URL grammar, e-mail addresses are found by the e-mail address grammar, and so on. Then, actions appropriate to each kind of object can be offered, supporting users in their work on those objects and on the document as a whole.

Our overall intent here – to examine document content, identify likely user actions, and provide simple ways of selecting and executing those actions – is not unlike that of the authors of other "intelligent" critic and advisory systems [e.g., 4, 6]. However, our work on structure detection differs from these systems in a number of ways:

- Syntactically-regular information structures, and the tasks that follow from them, can be found in almost any user domain. Hence, the total number of structures and tasks for which structure detection assistance would be helpful is too large for any single person or organization to try to satisfy. Therefore, we have paid special attention to the importance of allowing application developers and even end-users to define and extend the set of detectors and actions. This drove us to design

DOCKET ALARM Find authenticated court documents without watermarks at docketalarm.com.

a plug-in architecture for object recognizers and actions, as discussed in [7].

- We have remained open to large numbers of applications – ideally, any application available on the Macintosh platform – and have provided users with a consistent human interface across those applications.
- Viewing this work from an agent perspective, we have worked to keep the user well in control of the system's actions. This has called for a clear and explicit connection between the information that was found and the actions taken by users on that information. We have also kept the grain size of the actions relatively small, so that they would be easily understandable and, if need be, undoable.
- We wanted to keep our approach practical enough that it could ship as a commercial product, rather than simply being a research project.

Our most successful instantiation of the idea of structure detection (thus far) has been *Apple Data Detectors* (ADD): a system extension for Mac OS 8 that is now commercially available[1]. This first version of structure detection has been applied to the domain of Internet information management: finding structures like e-mail addresses, URLs, host names, and newsgroup names in user documents and automating actions on these structures, like creating a new e-mail message addressed to a discovered e-mail address or opening a web browser on a discovered URL. This capability has been implemented via Mac OS 8's *contextual menus*. Users select a region of text and activate the contextual menu by pressing the keyboard's Control key and the mouse button. This initiates the grammatical analysis of the selected text, and presents a hierarchical menu consisting of the structures found by the grammars and, for each of those structures, the actions that make sense for it. Users can examine all the discovered structures and their associated actions, and invoke whichever action they choose (Figure 1). More on the origin of Apple Data Detectors, its implementation, and how it made the transfer from research to product can be found in [7]. Some other related approaches to the structure detection question can also be found in [5] and [8].

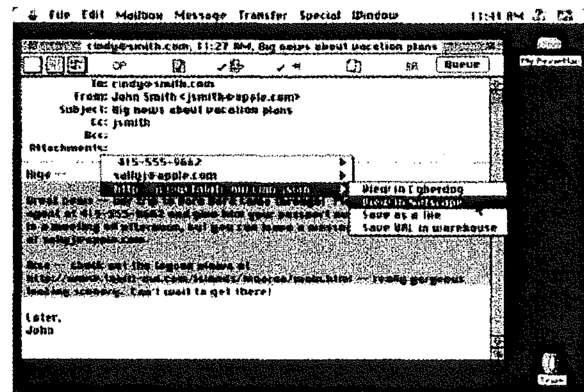

Figure 1: The results of invoking Apple Data Detectors on a text selection.

LiveDoc: Beyond Data Detectors

Many of the goals we had set for shifting the model of documents from passive streams of characters to manipulable collections of meaningful objects were met quite nicely by Apple Data Detectors. Nevertheless, we saw other opportunities for structure detection, growing out of both limitations of this first system and opportunities for expanding its breadth and depth.

Consider, first, the user interface to ADD. Users must select a range of text in a document, and invoke ADD through the contextual menu system. At that point, the selected text is sent to ADD's parser for analysis, which produces the menu of discovered structures and corresponding actions. This design has a number of implications on what kinds of interaction services can be offered to users:

- The basic premise of ADD is that it finds and selects manipulable bits of information for the user. However, in practice, the user must still find a structure that they would like to do something with, so that they can select, for analysis by ADD, a region of text containing that structure. It's true that ADD allows users to be inexact in selecting the region of text containing the structure of interest, since its grammars will find the desired structures in a stream of other, irrelevant characters. However, the tasks of discovering structures that might be operated upon and selecting the parts of the document around those structures are still imposed upon the user by ADD.
- It is not uncommon for useful structures to be nested within other useful structures. The URL "http://www.apple.com/default.html" contains within it the host name "www.apple.com"; a similar situation holds for the host name embedded in an e-mail address. These multiple structures can make the contextual menu generated by a typical text selection quite long, forcing the user to choose between limiting the set of active detectors (which keeps the task of finding the desired structure in the contextual menu a manageable one) and having to search through a large number of detected structures in the contextual menu (which makes ADD's services applicable in the largest number of situations).
- Listing the detected structures in the contextual menu doesn't really make them manipulable, through such direct manipulation techniques as drag and drop.

We encountered still other limitations resulting from our desire to increase the flexibility and power of ADD analyses. For instance, there really isn't any semantic interpretation of the discovered structures in ADD: Actions are associated with structures through a lookup table, not through any rich semantic representation of, for instance, what a URL is, what it might be used for, and what constraints exist on its use. As a result, the set of actions that can be offered to a user is fixed: it can neither include nor omit actions based on the semantics of the immediate interaction context. Further, since ADD's processing is tied to and activated by the contextual menu system, it

[1] as a a free plug-in for Mac OS 8; it is available via http://applescript.apple.com/data_detectors.

DOCKET ALARM Find authenticated court documents without watermarks at docketalarm.com.

must complete its analysis in the very short period of time in which users are willing to wait for a pop-up menu to appear[2] – about half a second. This is typically enough time for ADD to run a set of precompiled grammars and build a menu from a lookup table. However, it's easy to imagine more complex analyses of documents that could not be completed in this short amount of time.

Finally, we saw other opportunities for alternate analysis techniques, which could augment and extend the analysis model of ADD. ADD was built around a single parser, roughly equivalent to a context-free grammar (CFG); all structures found by ADD must be describable by a CFG. This is fine for simple structures such as phone numbers and e-mail addresses, but other more complex structures, such as a meeting announcement, are commonly found in a form not describable by a CFG, and so cannot be recognized by ADD. Some of these structures might be found by a more expressive grammar formalism, and others by recognizers that find structures by powerful but ad hoc analysis techniques instead of grammars. In addition, this textual grammar system cannot be easily extended to handle non-textual structures, such as images, drawings, circuit diagrams, or other kinds of visual information. All of these extensions indicate the need for a richer analysis model than was provided in ADD.

As a result, we began a follow-on project to ADD, known as *LiveDoc*, which would carry the ideas of structure detection forward to another level. In LiveDoc, the structure detection process is run in the background on the visible document's text, whenever that document is presented or updated. The results of LiveDoc's analysis are then presented by visually highlighting the discovered structures with a patch of color around the structure. Holding down a function key places the document in "LiveDoc mode" and presents the highlighted structures; releasing the function key returns the document to normal. Pointing at a highlight and pressing the mouse button then displays the menu of actions that can be applied to the structure, as shown in Fig 2.

Experientially, the design of LiveDoc draws on the Web in obvious ways: certain meaningful parts of a document are highlighted, and clicking on them causes certain actions to occur. LiveDoc differs from the Web, of course, in that we have substituted the automatic analysis of the document for the hand-authored links of web documents, so that any document in a LiveDoc-enabled application (more on this later) gains these characteristics. We are also able to assign more than one action to an object, something that lies outside the standard Web paradigm.

LiveDoc's use of background processing and automatic highlighting of discovered structures offers other advantages. Structures relevant to the user are automatically presented to the user while a document is in LiveDoc mode; interesting structures need not be searched for and highlighted manually. The problem of overly-long contextual menus is avoided, since a menu shows only the actions relevant to the structure it is associated with. Similarly, nested structures can be handled by nesting the mouse-sensitive regions around the structure: clicking on the host name part of a URL can present a menu of actions relevant to host names, while clicking outside the host name region presents actions relevant to URLs. This is shown in Figure 2: the host name of the e-mail address and URL ("sci.org") is shown with a darker highlight than those of the e-mail address and URL themselves. Finally, note that the visual representation of these structures means that, given appropriate software support, they can be treated as directly accessible components of the interface: they can take part in drag and drop interactions, and in other forms of direct manipulation.

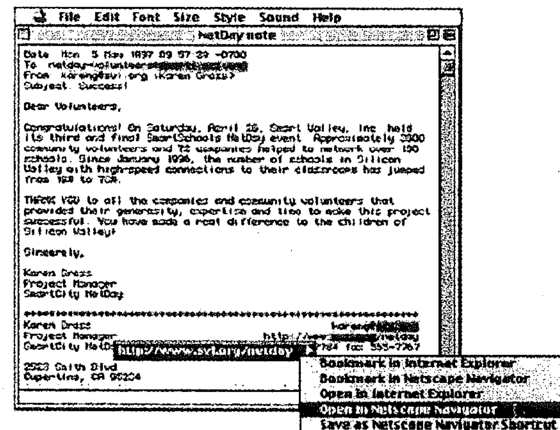

Figure 2: A sample interaction with LiveDoc. Note the highlighting of the discovered structures, the menu of actions available or the selected structure, and the nested highlighting of nested structures.

What is described above is, of course, only a general design for LiveDoc. To understand how this design can be implemented, it's necessary to look more closely at the system's architecture, and at its instantiation in several different working systems.

The LiveDoc architecture: A General Description

Architecturally, LiveDoc is built around the LiveDoc Manager (Figure 3). This component acts as an intermediary between the application making use of LiveDoc and the various internals of LiveDoc itself. In particular, the Analyzer System is made up of a set of detectors that analyze the content of the document passed to LiveDoc, a set of actions (typically, but not necessarily, implemented as AppleScripts) that carry out the various operations on the discovered structures, a table that specifies the mapping between detectors and actions, and an Analyzer Server that coordinates all these functions.

To make use of LiveDoc, applications must implement a small number of calls to the LiveDoc Manager, and a small number of

[2] In fact, the contextual menu system enforces a timeout after this amount of time: contextual menu plug-ins, like ADD, that have not completed their analysis when this time expires are halted by the contextual menu manager. When this happens, a "More actions..." option is added to the contextual menu; selecting this runs all the plug-ins to completion, and brings up a dialog box containing the results of their analyses.

DOCKET ALARM Find authenticated court documents without watermarks at docketalarm.com.

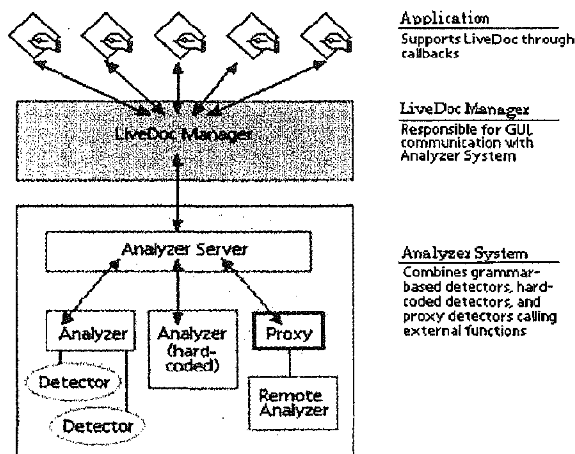

Figure 3: The high-level LiveDoc architecture

callback handlers to respond to calls from the LiveDoc Manager. The most important of these handlers inform the LiveDoc Manager of changes to the content of the document window, perhaps by the user's adding or deleting content, or by the scrolling or resizing of the window. The receipt of these calls by the LiveDoc Manager signals the Analyzer Server to analyze the text provided by the calling application; this will typically be the text currently visible in the applications' front-most window. Once the analysis is completed and the structures in the text have been identified, the LiveDoc Manager constructs the various highlights for the discovered structures and their corresponding menus of actions. LiveDoc knows where these structures appear in the text passed to it – an e-mail address might appear in characters 150 through 162 of the window's contents – but it has no idea where in the window those characters physically appear, and, thus, where the highlights should appear: this is information held by the application, not by LiveDoc. Hence, LiveDoc must ask the application for the information about the structures it has found via a callback. Once this information is available, the highlights and their associated mouse-sensitive regions can be constructed.

The LiveDoc Manager also controls the events that occur when the user presses the function key to enter LiveDoc mode, and when the mouse button is pressed while over a LiveDoc item. The LiveDoc Manager updates the display to present the highlight information over the discovered structures when the function key is pressed, and to remove the highlights when the function key is released. The LiveDoc Manager also receives the notification that the mouse button has been pressed over a highlighted item; it then gets the list of actions appropriate to the selected item and presents a menu of them to the user. If one of these items is selected, the action corresponding to the selection is run, producing the desired action.

Implementation 1: LiveSimpleText

Our initial explorations of LiveDoc were implemented in Lisp, primarily to gain a rapid understanding of some of the implementation issues we would ultimately face and to explore various human interface ideas. However, we soon needed to turn our attention to a system-level API for LiveDoc that would be both efficient and not terribly burdensome to developers who would have to modify their applications to take advantage of LiveDoc.

To test our first implementation of the LiveDoc API, we decided to modify a simple text editor application, SimpleText, to be a LiveDoc client. Although the API was designed as a Macintosh toolbox manager, so that it could be incorporated into existing applications without requiring developers to compile and link our code with theirs, this initial implementation did require linking portions of LiveDoc code with that of SimpleText. (This would be changed in later work where the LiveDoc manager was built as a shared library, allowing the LiveDoc Manager to be called by multiple LiveDoc clients.) We also needed a set of analyzers that could provide the document analysis services to the Analyzer Server. We decided to use the Apple Data Detectors context-free grammar engine, and to additionally implement a fast string search algorithm, as described by Aho and Corasick [1]; this analyzer rapidly finds all instances of strings in a document by comparing the text of the document to a set of dictionary entries. In doing so, we were able to test the multiple-analyzer part of the Analyzer Server and confirm its utility. As part of this test, we gave each analyzer its own interface affordance by varying the colors of the highlights: green for items found using a context free grammar (the Apple Data Detectors analysis engine) and pink for those items found using the string search.

The background processing of LiveDoc raises the issue of the proper way in which to update the display when changes are made to the document. Whenever the text in the window is changed, by either the system or the user, LiveDoc must re-analyze the text, since recognizable structures may have appeared or disappeared. But when should this analysis be done? It makes little sense to analyze a document while the user is working, since each change will require a re-evaluation of the text. Our current implementation works with a timeout, which starts a re-analysis when the keyboard has been inactive for a short period of time (about one second) after a change to the display. In this way, LiveDoc does not analyze the document while the user is typing, but resumes when there is a pause in the user's actions. We have considered various algorithms that might minimize the cost of this analysis – perhaps only analyzing the part of the window that had been scrolled, for instance – but our current implementation reanalyzes the entire content region of a window when a change is detected.

Overall, we believe there are some very compelling aspects of the LiveDoc interface as compared to Apple Data Detectors. As shown in Figure 2, LiveDoc displays its discovered structures in place, in the context in which they occur. It associates a menu of options with the object found where, in Apple Data Detectors, a single menu appears for all of the items found in the selection. Finally, LiveDoc works quietly in the background and displays the results of its analysis on demand, rather than performing the analysis on demand. Having said that, there are some user inter-

DOCKET ALARM Find authenticated court documents without watermarks at docketalarm.com.

face issues that we have not explicitly evaluated as part of this work, and that would benefit from further study. The use of highlighting is one of these: adding the notion of a sometimes-visible layer to the front of the display is a considerable change to the graphical interface, and, while others have pursued similar uses of translucency (e.g., [2]), its overall utility and understandability is worthy of study.

Similarly, the use of increasing degrees of saturation of the highlights should be examined: are these differences clear enough to be discernible, and do they require a precision in mouse control that users both possess and are willing to apply? Note also that this issue of manual and visual dexterity is complicated by the fact that using a function key as the means of entering and leaving LiveDoc mode means that we have made the interface a two-handed one: one for the function key and one for the mouse. This[3] raises disability and accessibility issues, and more general ease-of-use issues as well. Finally, we should note the current invisibility of the background document processing, and the need for the status of this processing to be visible to the user: How does a user know when the analysis is completed, and all the structures that can be discovered have been discovered? Some equivalent to a progress indicator bar, which reports how far along the analysis is, might be a useful addition. This may not be a problem when and if processors become fast enough to do these analyses very quickly, but we are hesitant to rely on sheer processor speed as the solution to what is really an interface design problem (especially since faster processors will most likely encourage developers to design more sophisticated, and probably more time-expensive, analyses).

Dealing with Static-ness: APIs and Plug-Ins

A second aspect of this project addressed some deeper questions about the LiveDoc architecture: the need for an API to the LiveDoc Manager (and whether it can be eliminated), the inflexibility of the analysis architecture, and the opportunities raised by a greater degree of semantic representation about the discovered structures.

LiveSimpleText worked well as a prototype, but we were still concerned about requiring developers to change their applications to gain access to LiveDoc's capabilities. We know from experience that developers are justifiably reluctant to change their applications just to implement a new feature provided by the toolbox, so we experimented with some alternatives that we hoped would ease this restriction.

Our first approach was to modify *TextEdit*, a set of Macintosh toolbox routines that provide an application with minimal text editing and display capabilities. Although TextEdit has a limit of 32,000 characters, too small for the needs of most modern word processing applications, most application dialog boxes that have editable fields use TextEdit, and some applications also use TextEdit for their editors or text display fields. If we could "trap" the calls from an application to TextEdit and pass them through LiveDoc, we could provide LiveDoc capabilities to any application that uses TextEdit, without requiring any modification to the application. Hence, we built a Macintosh system component that trapped and patched all the TextEdit calls, and then tested this with a variety of applications. In doing this, we discovered that most applications that use TextEdit take some programming shortcuts that kept our patches from working properly. We did however, find that the e-mail client Eudora used TextEdit appropriately, and we were able to provide LiveDoc functionality to an unmodified version of Eudora through this approach. Unfortunately, before we could deploy what we called "Eudora-Live", Eudora underwent an architectural change in moving from Eudora 2.0 to Eudora 3.0, and no longer used TextEdit.

A second promising approach was to use the framework of OpenDoc[4] as a vehicle for LiveDoc. At the time we undertook this work, OpenDoc parts and part editors were just beginning to appear, and it appeared that we could leverage some of OpenDoc's extensibility to implement LiveDoc as a plug-in to the OpenDoc architecture, so that OpenDoc text parts could acquire LiveDoc behavior without any source code modification. This was promising, but we ultimately discovered that it still was not possible to get information about the way in which text was rendered without the LiveDoc Manager consulting the application.

These attempts to build an API-less LiveDoc ultimately failed (as did other attempts not discussed here, which worked with other parts of the Macintosh architecture. Nevertheless, we are still optimistic about the possibility of such approaches to system extension, for LiveDoc as well as other kinds of extensions. In retrospect, it seems that what we were trying to do was to graft an object-oriented software model onto a platform that was not object-oriented. OpenDoc had the right sense of object-orientation, but its design happened to not expose certain aspects of the system that LiveDoc required. As operating systems continue to evolve in an object-oriented direction, we may find that the kinds of extensions we sought become the norm, rather than the exception.

Futures: Extensibility and Semantics

We tried to design LiveDoc's architecture to be as open as possible. In doing so, when faced with a tradeoff between performance and extensibility, we generally leaned towards extensibility. As implemented in the systems described here, there are two aspects to extensibility in LiveDoc: the analyzers and the kinds of structures they can discover, and the actions that can be taken on those structures. While the analyzers we built operated on lexical data, much could be done by applying the LiveDoc model to graphical information or multi-media content. We can easily envision analyzers that recognize features in drawings and pictures, or analyzers that "listen" for relevant structures in streaming audio or video. While we tried to be agnostic about these data types in the design of the LiveDoc architecture, it is inevitable that some pieces of the API would have to be rethought if such detectors became available.

The behaviors associated with actions should also be flexible and extensible, and work more closely with the document structure than they do at present. Our initial implementation of LiveDoc as LiveSimpleText assumed that actions would be handled by

3. Like the contextual menu system, it should be noted...

4. Cf. http://www.cilabs.com.

DOCKET ALARM Find authenticated court documents without watermarks at docketalarm.com.

DOCKET ALARM

Explore Litigation Insights

Docket Alarm provides insights to develop a more informed litigation strategy and the peace of mind of knowing you're on top of things.

Real-Time Litigation Alerts

Keep your litigation team up-to-date with **real-time alerts** and advanced team management tools built for the enterprise, all while greatly reducing PACER spend.

Our comprehensive service means we can handle Federal, State, and Administrative courts across the country.

Advanced Docket Research

With over 230 million records, Docket Alarm's cloud-native docket research platform finds what other services can't. Coverage includes Federal, State, plus PTAB, TTAB, ITC and NLRB decisions, all in one place.

Identify arguments that have been successful in the past with full text, pinpoint searching. Link to case law cited within any court document via Fastcase.

Analytics At Your Fingertips

Learn what happened the last time a particular judge, opposing counsel or company faced cases similar to yours.

Advanced out-of-the-box PTAB and TTAB analytics are always at your fingertips.

API

Docket Alarm offers a powerful API (application programming interface) to developers that want to integrate case filings into their apps.

LAW FIRMS

Build custom dashboards for your attorneys and clients with live data direct from the court.

Automate many repetitive legal tasks like conflict checks, document management, and marketing.

FINANCIAL INSTITUTIONS

Litigation and bankruptcy checks for companies and debtors.

E-DISCOVERY AND LEGAL VENDORS

Sync your system to PACER to automate legal marketing.

WHAT WILL YOU BUILD? | sales@docketalarm.com | 1-866-77-FASTCASE

fastcase®
Smarter legal research.