Drop Zones

An Extension to LiveDoc

Thomas Bonura and James R. Miller

Introduction

LiveDoc [6] is an extension to the Macintosh user experience that allows documents to reveal structured information in such a way that it can be readily identified and used to achieve specific actions. Various kinds of recognizers, including context-free grammars, are used to describe the structures to be found; these structures can be made up of either a single lexical term (either a variable structure like a phone number, or a collection of static strings, like company names) or multiple terms (for instance, a meeting can be defined as a combination of date, time, and venue structures). Small pieces of code can then be associated with each structure to instruct applications to carry out specific user actions on the discovered structures – perhaps to tell a telephony application to "Dial this phone number." These actions can then be offered to users by visually highlighting the discovered structures and attaching pop-up menus to the highlights. (See also [7] for an alternate interface formalism implementing the same notion of structure detection.)

This system can be very effective when working with structures that are simple and easy to recognize. However, some limitations to the approach have become clear in the time since our initial implementation of LiveDoc:

- *Limited interpretational power.* When a structure is difficult to characterize because of a high degree of variability in its form (such as the many different ways in which the information describing a meeting can be presented), the brittleness of LiveDoc's grammar-based analysis limits its effectiveness: LiveDoc may find some or all of the individual constituents of the meeting, but be unable to identify the composite structure because of where and how those constituents are located in the document. Unfortunately, this sort of structural complexity typifies much of the information for which people need computational assistance.
- *Too many choices.* There is a design tension implicit in LiveDoc: between encouraging developers to implement large numbers of actions for a given structure (so that LiveDoc can support as many user tasks as possible) and keeping the menus of actions small (so that the task of finding and selecting the desired action is kept simple). Ideally, these menus of actions should be kept small but relevant; the problem is that 'relevance' is really determined by the meaning of the context in which the user is currently working. For example, the name "Apple Computer, Inc." could be associated with such actions as, "Find the corporate headquarters on a map", "Get Apple's corporate phone number", "Get the current trading price of Apple stock", "Get the people in my address book associated with Apple" and so forth. All of these actions might be useful in one situation or another, but a user working on a financial task is far more likely to be concerned with the current price of Apple stock than the location of its corporate headquarters. Hence, the effective management of LiveDoc's action menus will come only with access to, and some understanding of, the contexts in which a user might be working.

Ultimately, both of these problems have their foundation in LiveDoc's lack of any sense of semantics of the objects on which it operates. The meaning of an object identified by LiveDoc is encapsulated within the grammar that recognizes it and the static list of actions that can be carried out on the structure: LiveDoc contains no explicit representation of this meaning. This makes it unwieldy, if not impossible, to define the mean-

ing of arbitrary sets of items or to describe anything about these structures that is not procedural. For instance, there is no way to capture abstract relations among structures, like the fact that people possess both telephone numbers and e-mail addresses.

Correcting these limitations required a different approach, one that would not only address the human interface concerns mentioned above but, more significantly, would position LiveDoc as an enabling technology for communicating with computational agents. This effort, called *Drop Zones*, is more than a new interface to LiveDoc. Rather, it is a framework centered on representing the meaning of LiveDoc objects, composing those objects might into other higher-level objects, and enabling users to take action on those compositions.

Drop Zones

A Drop Zone provides users with an interface for managing LiveDoc objects in the context of a set of typical user tasks. Most importantly, the underlying framework for Drop Zones allows for the explicit definition of semantics about the objects, separated from the grammars that define the objects as textual strings. Once this has been done, a user interface to these objects and their semantic interpretations can be provided.

An interaction with the Drop Zone interface is shown in Figures 1 and 2. The window named 'test' in Figure 1 belongs to a LiveDoc-enabled word processor, *LiveSimpleText* (see [6]), and shows a number of structures within the document in view having been recognized by the analyzers. The window labeled *Activities* is a Drop Zone interface to a set of interpreters or 'assistants'. Each of these assistants, *E-mail, Telephony, Finance* and *Appointment*, implements a knowledge base that can operate on appropriate sets of LiveDoc structures. These assistants make their capabilities visible when the user selects various structures identified by LiveDoc and drags them to the assistants.

The important aspect of the Drop Zone interface is that it allows the user to work with the objects of interest in specific, understandable contexts. Simply working with the semantics of a set of individually meaningful objects, such as a personal name, a time and a telephone number, is too open-ended to permit much useful assistance to the user: there are too many ways in which these objects might be combined. However, thinking about the name of a person and a phone number from the perspective of making a telephone call easily leads to the interpretation, 'Call this person at this number'. Similarly, thinking about this information from the perspective of an address book easily leads to the interpretation , 'Add this person to my address book'.

Drop Zones offer a way to make these contexts a tangible part of the user interface. A Drop Zone assistant can be thought of as an interpreter that takes features identified by LiveDoc, interprets their meaning with respect to its context, and recommends appropriate actions. Consider Figure 2, in which the user has selected the structure Tom Bonura, which LiveDoc has identified with its personalName recognizer. When an object is selected, it is sent to the Drop Zone control system. Each of the assistants determines if it is able to accept and act upon the set

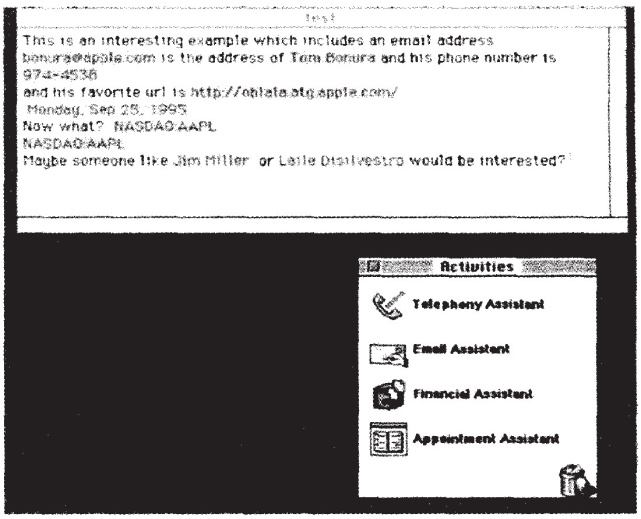

Figure 1: Drop zone is shown in the window labeled 'Activities'. The window at the top called 'Test' is a LiveDoc window showing proper names, e-mail addresses phone number, URL, date and stock market ticker codes

of currently selected objects. Any assistant that can do something meaningful with the objects will, when asked, highlight itself in a manner consistent with the Macintosh Drag Manager (typically, by drawing an enhanced border around itself when the mouse is above it). In this case, only the E-mail Assistant can accept the name of a person (that is, an object of type personalName), and so it indicates its availability to the user by drawing a highlight rectangle around itself when the object being dragged is over it.

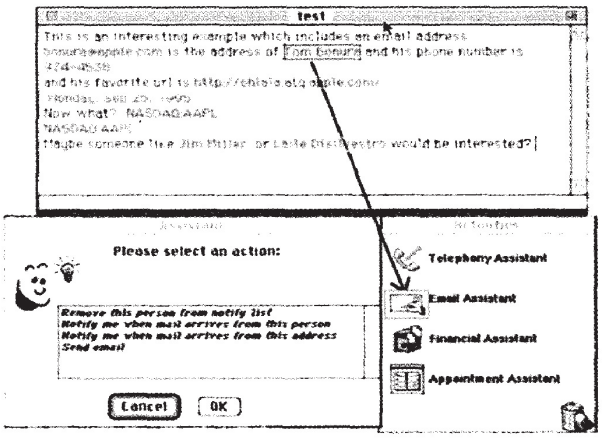

Figure 2: A user interaction with Drop Zones

After the user drops the name on the E-mail Assistant, a set of actions that make sense for people are presented in the *Assistant* window, and the user can choose among them. In this case, there are four possible actions, the first two of which operate on people's names and the last two of which operate on people's e-mail addresses. What is interesting in this example is that the e-mail actions are available, even though the object being offered to the assistant is a personalName. The E-mail Assistant's ability to do this is based on its access to a body of semantic information about the various types of objects present in the system, and its own design to provide e-mail-oriented assistance to the user.

Other interfaces to these assistant capabilities are, of course, possible. In particular, it is easy to imagine modifying the LiveDoc menus so that these inferred actions are presented along with those directly associated with the object clicked upon. The risk here – and the reason that we chose the alternative shown in Figure 2 – is that, as the number of selected objects and the size of the system's knowledge base increases, there may be an explosion of inferred actions expanding the size of the object's action menu beyond manageability. It's worth noting that Dey, Abowd, and Wood [3] have taken a more expansive approach to this problem, and include any action that can be inferred from any object. It will be interesting to see how well their design choice works under the circumstances described here.

Semantics and Representation

As mentioned above, objects identified by LiveDoc have no explicit semantic value, other than their intrinsic meaning to the user of their names. It is easy to imagine the kinds of assistance that could be offered by a system that knew things like 'an important meeting is one that is called by anyone in your management chain' and 'you should always accept invitations to important meetings.' However, knowledge like this can't be stated in the simple kinds of lexical grammars used by LiveDoc. A major part of the design of Drop Zones is then to represent semantic information like this – of, about, and related to the structures being recognized by LiveDoc – in a knowledge base. The current implementation uses a hybrid knowledge representation language built around a frame-based object system, augmented by relational axioms to express facts about these objects. In this way, we keep object semantics separate from syntax (i.e., the grammars used for structure recognition in LiveDoc), and maintain the flexibility to assert facts unrelated to the actual objects found in the user's document (like the 'important meeting' example above).

When objects are selected, they are inspected by the assistants in the Drop Zone. These assistants are built around a collection of facts and axioms that determine whether and how they can operate in some meaningful way on various kinds of objects. These facts and axioms, and the inferences they permit, allow Drop Zone assistants to make much richer interpretations of structures and offer much more relevant actions than does LiveDoc's rigid model of structure grammars and static lists of actions.

Consider the situation in which a user drags a telephone number to the E-mail Assistant, presumably so that the user can send an e-mail message to the person who possesses that phone number. A literal examination of the object reveals nothing of use to the assistant: sending e-mail messages requires an e-mail address, not a phone number. However, as part of its design as an assistant for e-mail tasks, the E-mail Assistant includes an axiom that can derive e-mail addresses from phone numbers, by finding a person with the given phone number, and then obtaining the e-mail address of that person. That is, it needs to use the phone number passed to it to unify the expression:

```
(and
 (PHONE-NUMBER
    ?Person ?ThePhoneNumber)
 (e-mail-ADDRESS
    ?Person ?TheAddress))
```

and to produce the binding of the appropriate e-mail address to the variable ?TheAddress.

The problem now becomes: Where does the E-mail Assistant find the set of people whose phone numbers and e-mail addresses can be examined? The Drop Zones representational system provides two ways through which an assistant can gain access to this information. Mappings can be built between the objects inside the Drop Zones representational system (e.g., there is an object called PERSON, which has such attributes as PhoneNumber and EmailAddress) and databases or other applications. Such a mapping, in combination with a scripting language or some other programmatic way of manipulating applications, enables the E-mail Assistant to look inside an address book application for a person with the stated phone number. Another call to the address book application, guided by another mapping rule, will return the e-mail address for the identified person. Alternatively, these facts can be held directly within the Drop Zones representational system, as relations of terms, such as:

(PHONE-NUMBER 'Tom Bonura' 974-4538)

These terms can be provided by developers or, through an appropriate human interface, end-users. As a result, these bits of information can be provided to the system as they are needed, and, by preserving the assistants' knowledge bases across invocations, made available to future uses of the assistants.

Needless to say, this part of the Drop Zones work runs directly into the historical problems of knowledge representation and knowledge acquisition. The difficulty of this problem should not be underestimated, especially given its importance to Drop Zones. However, the complexity of this problem as it relates to Drop Zones is limited by the narrow scope of the assistants. The bodies of knowledge relevant to a typical assistant are small, and relatively self-contained. The representational task is then much more like that of a well-constrained expert system [e.g., 4] than a large and open-ended knowledge system [5], where the problems of representational consistency and inter-

connectedness of different knowledge units are both critical to the system's success and extremely difficult to resolve.

Composing Terms

A representation of semantics can be useful in other ways. Consider the problem of trying to invoke an action on a collection of differing terms in a document. For instance, the sender of the e-mail message in Figure 3, based on a human's reading of the message, is clearly interested in getting together for lunch. LiveDoc has identified a number of structures in the e-mail message, which, as usual, are shown with colored highlights. However, the bits of information that make up the details of this proposed meeting are spread throughout the message, and, as a result, they fail to match the rigid syntactic 'Meeting' grammar built into LiveDoc.

Drop Zones goes beyond LiveDoc in allowing the user to select some subset of those terms and drag them as a group to the meeting assistant for interpretation. Because the Meeting Assistant contains an axiom specifying that a meeting is indicated by a date, a time, a venue, and a person's name, this collection of objects is recognized as a meeting by the Meeting Assistant. This assistant will then highlight itself when the objects are dragged over it, and an action like 'Add this meeting to your calendar' can be offered to the user via the Assistant window.

Communicating with Agents

The concept of an 'intelligent agent' has been an active research topic in recent years, although problems still hamper the development and deployment of agent-based systems. One of the most serious problems faced by these systems is how to enable users to communicate with them. Task delegation to an agent is by its very nature an ambiguous undertaking; if it were not, we could hardly call it delegation. Users must describe their intended task to the agent, which must then devise a means of addressing it (or determine that addressing it is beyond its abilities), and communicate back to the user what it has done. Throughout this process, of course, the user needs to be aware of what the agent is about to do, what it is currently doing, and what it has just done. The user should also be able to interrupt the agent at any stage of its action, and either prevent it from taking some action or undo an action that has been taken.

To this end, many researchers have explored systems that are activated by recognizing patterns in user behavior and offering to complete the user's intended actions [e.g., 1]. Such approaches avoid direct communication with the agent, in the hope that the agent will be able to recognize an action worth taking, without explicit direction by the user. The risk here, of course, is that an agent will misrecognize or fail to recognize a pattern in the user's behavior. Other approaches use some limited form of natural language understanding as an agent communication language (e.g., [2]). The viability of this approach is limited by the complexity of the natural language understanding problem and the difficulty of extending it to all the possible problems that a user might wish to address. Users are additionally challenged by, in most cases, having a limited understanding of what kinds of actions the agent can take on

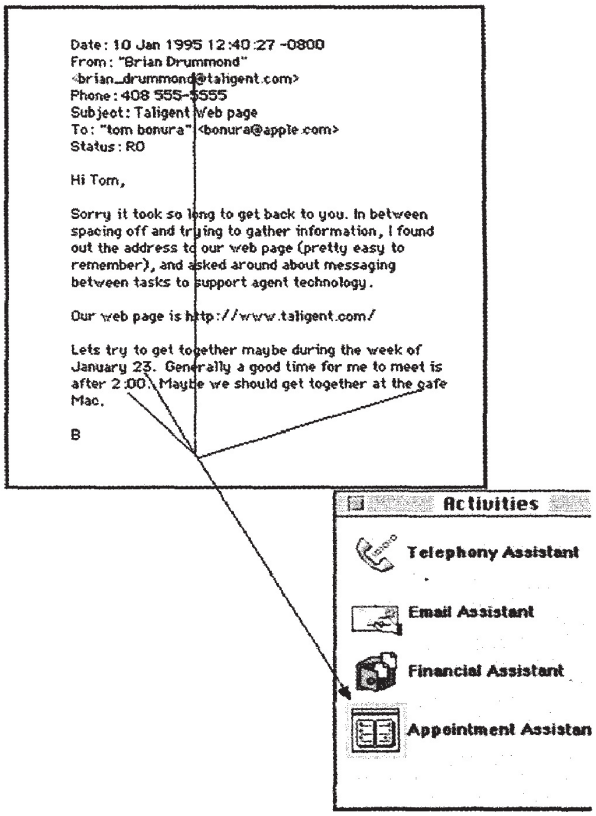

Figure 3: Selecting multiple items from a LiveDoc window for interpretation

their behalf, and of what kinds of language terms and constructs they can use in communicating with the agent.

We have designed the Drop Zone interface as a way of communicating intent to an agent by specifying the kinds of objects that need to be manipulated and the context in which they should be interpreted. The system provides feedback on this object selection, indicating whether it can operate on the selected items; it also provides feedback indicating what services it can provide. The ambiguity and open-endedness of a natural language interface is avoided, and a clear picture is given to the user of what services the agent can provide to the user, before those actions have been taken. Direct manipulation interfaces have often been characterized as in some way antithetical to agents (see [8]); Drop Zones shows how the interactional strengths of direct manipulation can serve as a gateway to the delegational ability of intelligent systems.

Conclusion

The Drop Zone architecture is a powerful extension of LiveDoc's capabilities. It provides a direct manipulation interface for specifying actions to be taken on terms identified by Live-

Doc, and for selecting and confirming the actions desired by the user. Drop Zones use knowledge representation and limited inference to determine what actions users might profitably apply to various types of objects under different interaction contexts. They provide users with clear feedback indicating when an assistant can operate on the selection and what actions can be applied to the selection. What is especially significant about this approach is that, since knowledge can be added to the system dynamically, as either new facts or complex axioms, the behavior of the system is highly flexible and adaptable. We have spoken here and elsewhere [6] of the need to move to a more sophisticated model of documents, one that views documents as collections of meaningful objects rather than simple streams of characters. The combination of LiveDoc and Drop Zones offers a significant step in that direction.

References

1. Benyon, D., and Murray, D. (1993). Developing adaptive systems to fit individual aptitudes. *Proceedings of the 1993 International Workshop on Intelligent User Interfaces.* Orlando, Florida.
2. Cohen, P., Cheyer, A., Wang, M. and Baeg, S. (1994). An open agent architecture. O. Etzioni (Ed.), *Proceedings of the AAAI Spring Symposium Series on Software Agents,* (Stanford, California, March 1994). American Association for Artificial Intelligence, p. 1-8.
3. Dey, A. K., Abowd, G. D., & Wood, A. (1998). CyberDesk: A Framework for Providing Self-Integrating Context-Aware Services. *Proceedings of Intelligent User Interfaces '98.* New York: ACM Press.
4. Hayes-Roth, F. (1983) *Building Expert Systems (Vol. 1).* New York: Addison-Wesley.
5. Lenat, D. B., & Guha, R. V. (1990). *Building Large Knowledge-Based Systems: Representation and Inference in the Cyc Project.* New York: Addison-Wesley.
6. Miller, J. R., & Bonura, T. (1998). From documents to objects: An overview of LiveDoc. *SIGCHI Bulletin,* this volume.
7. Nardi, B. A., Miller, J. R., & Wright, D. J. (1998). Collaborative, programmable intelligent agents. *Communications of the ACM,* Vol. 41, No. 3 March,1998.
8. Shneiderman, B., & Maes, P. (1997). Direct manipulation vs. interface agents. *interactions,* 4(6), pp. 42-61.

About the Authors

Thomas Bonura is a Senior Scientist at Apple Computer, Inc. currently working on applications of speech technologies to the Macintosh user experience. His research interests are in user interface design and implementation and knowledge based systems.

Jim Miller, until recently, was the program manager for Intelligent Systems in Apple's Advanced Technology Group. He is currently exploring consumer applications of Internet technology as part of *Miramontes Computing.*

Authors' Addresses

Thomas Bonura
Apple Computer, Inc.
1 Infinite Loop, MS 301-3KM
Cupertino CA 95014
bonura@apple.com or bonura@acm.org

Jim Miller
Miramontes Computing
828 Sladky Avenue
Mountain View CA 94040
email: jmiller@millerclan.com
Tel: +1-650-967-2102