Bonnie A. Nardi, James R. Miller, and David J. Wright

Collaborative, Programmable Intelligent Agents

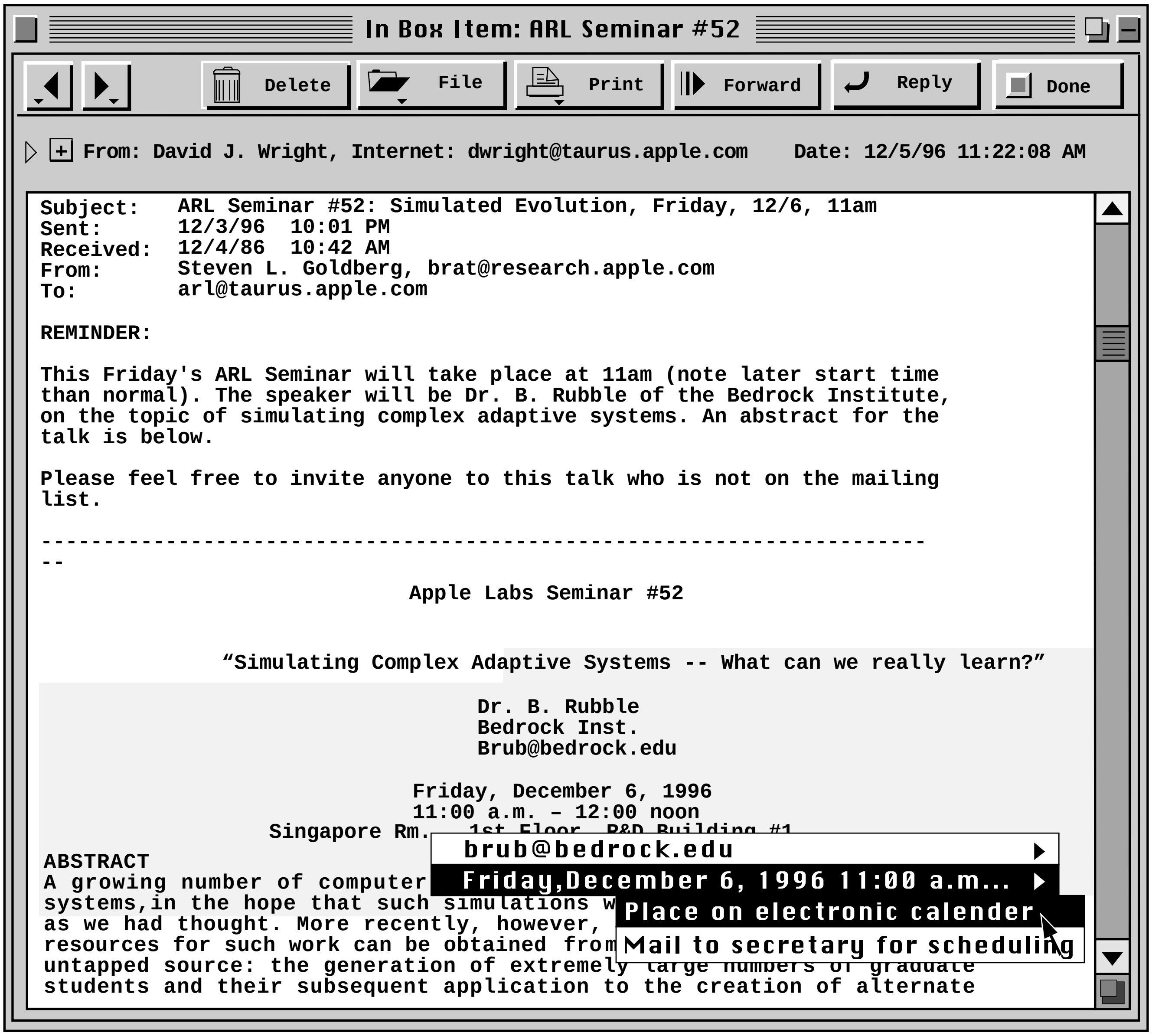
Extracting semantics from everyday documents, intelligent agents, illustrated by Apple Data Detectors, infer high-level goals from simple user actions.

We began our research on intelligent agents with the same romantic imagery that has always fueled interest in agents: Robbie the Robot from "Forbidden Planet," HAL from "2001: A Space Odyssey," the Star Trek computers (including Data). The images they conveyed of intelligent machines and the ways people would interact with them—by talking as if to an old friend who knows you so well they could finish your sentences for you—were so attractive that a whole generation viewed them as proven technology. They also became the models for the intelligent interface community that emerged out of artificial intelligence research in the 1980s, even for early work on AI itself.

But building anything approximating real intelligence into a computer has proved to be a painfully difficult task, and the powers of Robbie and HAL have remained beyond our grasp. We need to step back a bit, think carefully about what people and computers are each good at, understand how they can complement each other, and where we as system designers can do some good.

Shneiderman [12] observed that claims about intelligent software agents are vague, dreamy, and unrealized. As Apple Computer researchers, we started from a simple but focused approach to agents: That they should have the ability to infer appropriate high-level goals from user actions and requests and take action to achieve these goals. Further, based on a study of reference librarians as exemplary human agents [9], we wanted to build a system in which the user would not have to state goals explicitly and in detail. We learned from librarians that a large part of their value to clients is in working with imprecise requests. Beyond this concern, our general design strategy was to keep the most basic user question in front of us at all times: Will this software do something useful for users in an intelligent way that makes them more productive? The system we describe here—called Apple Data Detectors—meets our criteria of being unobtrusive, being able to infer user needs, and doing useful work. Apple Data Detectors shipped as a product in 1997.

Earlier work on intelligent agents was multi-


In Box Item: ARL Seminar #52
Delete
File
Print
Forward
Reply
Done
brub@bedrock.edu
Friday,December 6, 1996 11:00 a.m...
Place on electronic calender
Mail to secretary for scheduling

```
From: David J. Wright, Internet: dwright@taurus.apple.com    Date: 12/5/96 11:22:08 AM

Subject:   ARL Seminar #52: Simulated Evolution, Friday, 12/6, 11am
Sent:      12/3/96  10:01 PM
Received:  12/4/86  10:42 AM
From:      Steven L. Goldberg, brat@research.apple.com
To:        arl@taurus.apple.com

REMINDER:

This Friday's ARL Seminar will take place at 11am (note later start time
than normal). The speaker will be Dr. B. Rubble of the Bedrock Institute,
on the topic of simulating complex adaptive systems. An abstract for the
talk is below.

Please feel free to invite anyone to this talk who is not on the mailing
list.

-----------------------------------------------------------------------
--
                         Apple Labs Seminar #52


            "Simulating Complex Adaptive Systems -- What can we really learn?"

                             Dr. B. Rubble
                             Bedrock Inst.
                             Brub@bedrock.edu

                         Friday, December 6, 1996
                         11:00 a.m. - 12:00 noon
               Singapore Rm., 1st Floor, R&D Building #1
ABSTRACT
A growing number of computer
systems,in the hope that such simulations w
as we had thought. More recently, however,
resources for such work can be obtained from
untapped source: the generation of extremely large numbers of graduate
students and their subsequent application to the creation of alternate
```

Figure 1. Sample invocation of Apple Data Detectors. The user has selected a portion of an email message describing an upcoming seminar. Two patterns are found: an email address (brub@bedrock.edu) and the announcement of the meeting (the sequence of date and time information starting Friday, Dec. 6, 1996). These patterns are presented in the pop-up menu; by pointing at the date information in the menu; a second pop-up menu offers a choice of actions: place an entry for the meeting on the user's electronic calendar or mail the selection to the user's secretary. The user can select one, thereby running a small application, or move the cursor off the menu, eliminating the pop-up menu and canceling any actions.

faceted, to the point where it is difficult to find a consensus among researchers on exactly what constitutes an "agent" or even "intelligence." However, in nearly all cases, systems described as "agent-based" rely on some explicitly represented knowledge about relevant aspects of the world—the objects or concepts being addressed by the software, the tasks relevant to the user, and the user's own knowledge about the world. Researchers have used machine learning techniques to track user actions and construct models of user preferences [7], create explicit models of user knowledge and skill levels in an attempt to anticipate user actions, misconceptions, and information needs [2], and implement planning systems to leap from a user's stated intention to the specific actions required to achieve that intention [3]. The locality of

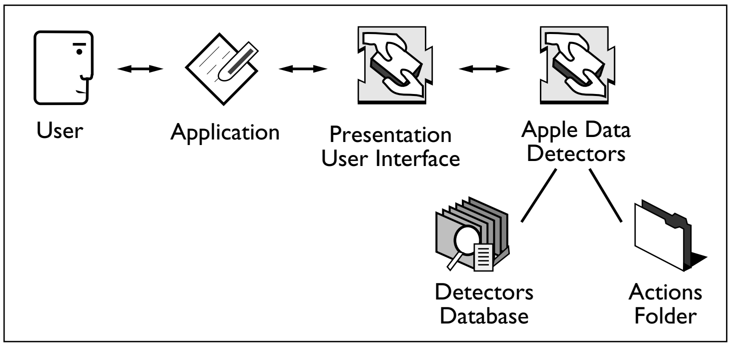


Figure 2. The Apple Data Detectors architecture, which separates the application in which the information is found, the presentation of the analysis and possible actions, and the analysis of the information itself. This separation means that Apple Data Detectors can be invoked in any application and that the user interface can be implemented, refined, and evolved separately from the analysis module.

```
HTTP = (Http Protocol, Host, Port?, Path?, {Http Location, Http Search} ? )
Http Protocol = {"http://", "https://"}
Port = (":", Port Number)
Port Number = Digits
```

Figure 3. A grammar to define a URL. The language implements a context-free grammar in which sequences of terms are matched against the input stream. References to other grammars are permitted, as are optional and repeated terms. Here, the HTTP grammar finds a match when it finds an HTTP protocol, a host, a port (optional), a path (optional), and either an HTTP location indicator or an HTTP search command and arguments.

agents also varies across different agent-based systems; some act only within one's own machine, find others autonomously crawl the Web, searching for interesting content [4]. We tried to find a middle ground by using explicit representations of user-relevant information as a means of identifying actions users might wish to take but to leave the choice of these actions to users.

Working with Information Inside User Documents

Our first step was to find a user problem that needed solving in which intelligent agents would add value. In an investigation of how people file information on their computer desktops [1], we discovered that a common user complaint is that they cannot easily take action on the structured information found in everyday documents (structured information being data-recognizable by a grammar). Ordinary documents are full of such structured information: phone numbers, fax numbers, street addresses, email addresses, email signatures, abstracts, tables of contents, lists of references, tables, figures, captions, meeting announcements, Web addresses, and more. In addition, there are countless domain-specific structures, such as ISBN numbers, stock symbols, chemical structures, and mathematical equations. These structures are not only relevant to users, but because of their structure, are also recognizable by parsing technologies. Once identified, the structure's type can be used to identify appropriate actions that might be carried out, like placing a meeting on a calendar, adding an address to an address book, dialing a phone number, opening a URL, finding the current price of a stock, filing an ISBN number, and compiling a list of abstracts.

Apple Data Detectors supports a wide range of uses. Think of all the structured information in the documents you work with; in addition to those mentioned already, add bibliography items, forms (such as travel expense reports and non-disclosure agreements), executive summaries, and most important, such domain-specific kinds of data as legal boilerplate, customer orders, and library search requests. Specific detectors can be created for each of these types of information.

User interface. To use Apple Data Detectors, users select a region of a document with some information of interest. Pressing a modifier key and the mouse button instructs the system to analyze the data within the selected region and to find all structures for which it has grammars. It then offers appropriate actions for each structure (see Figure 1). For example, for users reading email who come across a seminar announcement they would like to put on a calendar, Apple Data Detectors parses the relevant information within the selected text, including the meeting's place, time, and date and puts this data into the appropriate fields on the calendar.

A user can select a whole document or part of a document without having to make a careful selection; the grammars find any embedded structures they know about within the selection and offer an appropriate set of actions from which to choose.

The use of anthropomorphism in an agent interface [12] was incongruent with our goal of unobtrusiveness. We designed Apple Data Detectors to be invis-

```
–addressLetterTo: Given a phone number, find the person with that phone number and
–address a letter to him/her.

on addressLetterTo(phoneNumber)

        –Open Now Contact and find a person with the indicated phone number
        tell application “Now Contact 3.5”

                --find the person with the supplied e-mail address
                set thePerson to the first person whose (work phone is phoneNumber)

                --get the address information for this person
                set firstAndLastName to (the first name of thePerson) & “ ”
                        & (the last name of thePerson)
                set theAddress to firstAndLastName & return & (the company of
                thePerson)
                        & return & (the work address of thePerson) & return
                        & (the work city of thePerson) & “,” & (the work state of
                the Person)
                        & “ ” & (the work zip of thePerson) & return & return
        end tell

        --Open WordPerfect and write the address information into a new document
        tell application “Corel WordPerfect”
                –open an empty piece of WordPerfect stationary
                open (path to system folder as string) & “DD template”

                –get today's date
                set theDate to (the current date as string) & return & return

                –get the salutation: something like “Dear John,”
                set theHeader to “Dear” & the first word of firstAndLastName & “, ”
                        & return & return

                –write all of this as a new paragraph at the beginning of the document
                make paragraph at the begining of the document with data
                        theDate & theAddress & theHeader
        end tell
end addressLetterTo
```

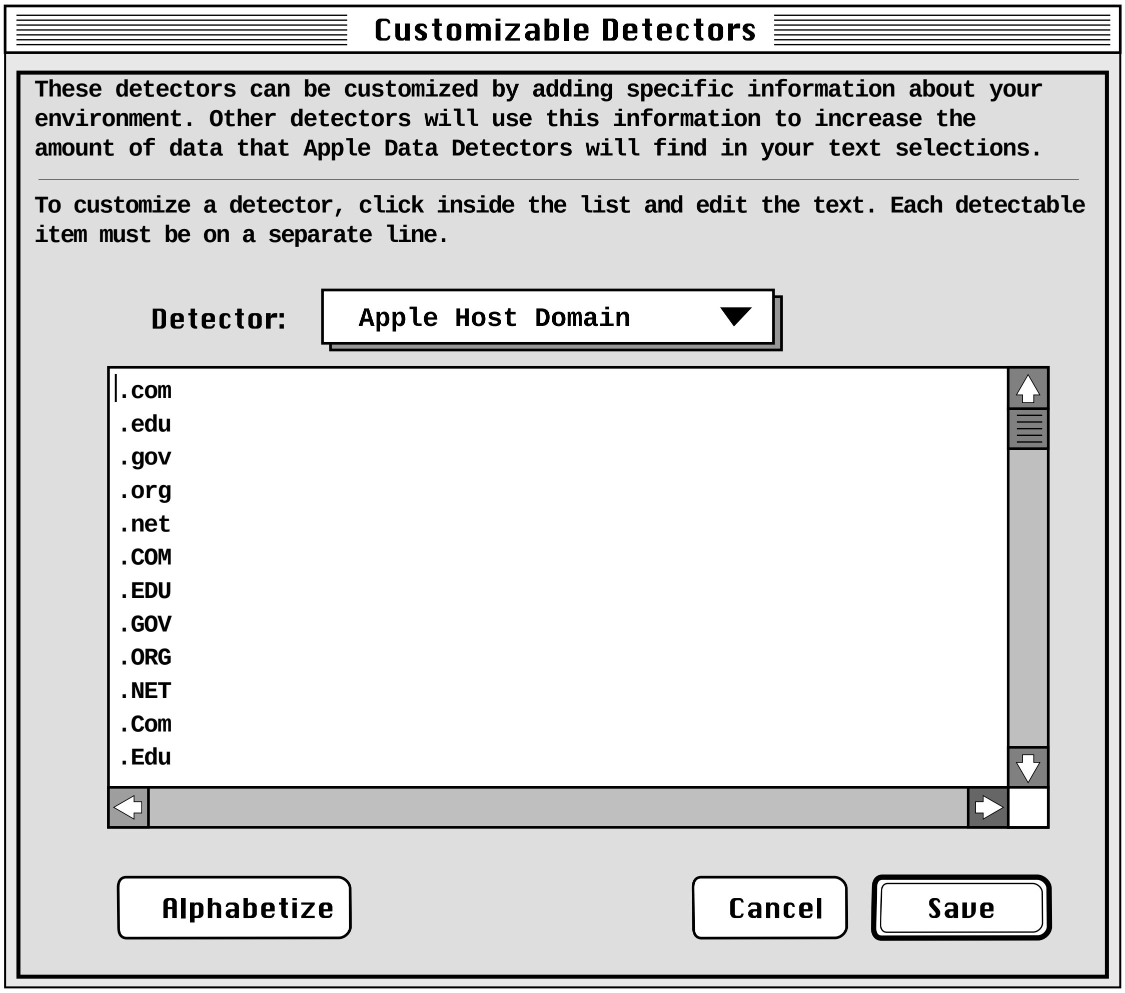
Figure 4. An action script, demonstrating the generality of Apple Data Detectors' use of a scripting language and external applications as information repositories and as end-user tools. This script can be activated when the system detects a telephone number. It then generates word processor letterhead addressed to the person possessing that number, with appropriate date and salutation information. This script uses two applications: First, a “personal information manager” (Now Contact 3.5) is opened and used as a database. Then the script opens an empty word processor document (via Corel WordPerfect) and writes the date, name, address, and salutation into it, leaving the user ready to write the letter.

ible until needed (a butler only when you want a butler). Thus there is nothing like a “swivelling eyeball” watching the user as in the Selection Recognition Agent [10] or a character as in Microsoft's Bob. Apple Data Detectors acts behind the scenes, emerging only when summoned.

Architecture and implementation. Apple Data Detectors is an open extensible system allowing the recognition and parsing of complex structures. Its recognition technology is a hybrid system that uses Earley's algorithm and deterministic finite automata. The algorithms permit recognition of not only simple structures, such as predefined strings and atomic patterns like URLs and email addresses, but complex composite structures, such as meeting announcements composed of smaller more atomic structures, like date, time, and place. The idea of detecting structures is not new [11], but our work is unique in providing an open system for creating complex new structures and actions. It is not difficult to hard-code a recognizer for an atomic structure, such as a URL, but substantial work is still required to craft an architecture that opens up the process of creating complex structures.

To make Apple Data Detectors work, grammars (we call them “detectors”) and action scripts have to be written (see Figures 2–4). Today this task is for programmers only, though it's made as easy as possible through the use of a special-purpose editor and AppleScript. We are committed to providing facilities for end users to create their own detectors and actions and have begun work in this area. While Apple and third-party developers will provide many detectors and actions, it is clear that enabling end users to write their own detectors and actions will make Apple Data Detectors much more powerful and useful by providing domain-specific programming capabilities appropriate for the specific needs of specific users [8].

The system parses the selected text according to the grammars associated with them. For each structure found by a detector, a data record is produced describing the structure. This record can then be passed to an action script for execution, in much the same way a subroutine is invoked with a specific set of parameters. These parameters depend, of course, on the kind of structure found by a given detector.


Customizable Detectors

These detectors can be customized by adding specific information about your environment. Other detectors will use this information to increase the amount of data that Apple Data Detectors will find in your text selections.

To customize a detector, click inside the list and edit the text. Each detectable item must be on a separate line.

Detector: Apple Host Domain

```
.com
.edu
.gov
.org
.net
.COM
.EDU
.GOV
.ORG
.NET
.Com
.Edu
```

Figure 5. User customization window in which users can inform Apple Data Detectors about information relevant to their environments.

Detectors for strings and atomic patterns typically create a record containing only the structure that was found—such as the name of a conference room or an email address. Detectors for complex patterns, such as a meeting announcement, produce records containing each of the components playing a part in the recognition of the pattern. Note that a detector can have an action associated with it and also play a part in a more complex detector. For example, a conference room detector could have a "Show on map" location indicating where that room is located and also play a part in defining the more complex meeting detector.

End users can also specify simple detectors in the form of lists of strings, such as lists of conference rooms, group members, and customers (see Figure 5). These user-specified detectors can then be used in other detectors, increasing the personalization of the system. For example, the meeting detector might refer to the conference room detector, which could be written as a list of strings by an end user or by a local systems administrator. The meeting detector could then be used in a number of organizations without requiring the meeting detector developer know the name of every conference room in every organization that might use it.

The Apple Data Detectors' architecture separates detectors and actions so that more than one action can exist for any detector (without having to duplicate the detector for each action). Hence, a detector written by one person to support one task can be used by another detector for another task. Detectors are thus reusable and easily shared.

Detectors also should be shareable for the sake of compatibility. The "place meeting on electronic calendar" action might expect the fields "StartDate," "StartTime," and "EndTime" from the "Meeting" detector. If a new definition of "Meeting" is installed that does not export these fields, the actions currently installed would not work properly. We added the notion of namespaces to detector definition, so these definitions can be merged and extended by placing them in the same namespace or kept separate by placing them in different namespaces. Authors of actions must then observe these namespaces, so their actions are assured of receiving the arguments they are intended to receive.

Apple Data Detectors inserts a new kind of programming capability into the middle of user interaction. Unlike conventional scripting, Apple Data Detectors works hand-in-glove with the data in users' applications and interactively with users as they work. The job of supplying data to the parameters of the scripts is taken care of by users simply making a selection containing the data—without having to leave the application, type anything, understand order of parameters, or know any other low-level programming details needed in conventional scripting.

Unlike systems based on predefined recognizer/action pairs, such as the Selection Recognition Agent [10], the Apple Data Detectors' scripting ability allows any set of arbitrary actions to be executed when the user selects a particular action (see Figure 3). Scripting is not limited to the parameters of a command line interface to the application; it can do anything that can be expressed in the scripting language, including manipulation of data structures inside the application, if the application's scripting model makes that possible.

Because Apple Data Detectors is a general-purpose programmable engine (not merely a collection of specific detectors and actions), it has the potential to transform the way users work. Without the need to modify data by changing it into objects, database-readable data, or any other complex format, a powerful new capability is introduced into the system. Any application can provide information to Apple Data Detectors, and any scriptable application can respond to Apple Data Detector actions without having to change in any way. The ability to work within existing documents provides immediate user value and leverages the data the user is already using. Structures in other formats, such as objects (in the object-oriented sense), are amenable to the system's parsing and scripting capabilities, representing a flexible technology that can evolve and grow with changes in data formats.

Apple Data Detectors assumes a world of heterogeneous data (in the user's machine) that comes from different applications in different file formats. For example, it makes sense to put an address in an address book, whether the address is from a message sent in any of several mail programs, appears in a downloaded document, or is in another address book maintained by the user. Apple Data Detectors is a pervasive technology, giving users access to actions appropriate for data in an entire set of documents.

Collaborative vs. Autonomous Agents

The system's current user interface allows users to select data and actions, resulting in a collaborative agent. The user participates by signaling as structures of interest occur in a document and by verifying that an action is appropriate by selecting it from the menu, as shown in Figure 1. Apple Data Detectors participates by recognizing structures, offering appropriate actions, sending data to the target application, opening the target application, and performing any other actions specified in the action scripts.

Apple Data Detectors therefore has the ability to infer appropriate high-level goals from user actions and requests and take appropriate action to achieve these goals. When users invoke it on a region of text in a document, they are saying, in effect, "Find the important stuff in here and help me do reasonable things to it." Users can be imprecise, throwing the system a broad hint that there is something of interest, then let the system use its knowledge to do the right thing. Users work on their tasks in terms of high-level goals, such as "put this address in my address book"—not by opening folders, clicking on icons, cutting, and pasting. Direct manipulation is a wasteful, frustrating way for users to interact with machines capable of showing more intelligence.

Having to choose a particular action is actually an artifact of Apple Data Detectors' ability to find more than one structure in a selection and the need to offer more than one action for the same structure (such as "open URL" and "place URL in hot list"). But multiple structures and actions are of benefit to users in terms of their being able to choose the structure to be manipulated and to choose the action to be employed. Users therefore remain in control of their work with the computer at all times.

Developer and user interest. The response of users and application developers to Apple Data Detectors has been extremely positive. The code required to implement detectors and actions is small enough that developers are eager to link their applications to the technology. Users see it as a valuable way to rid themselves of annoyingly detailed interaction. At the same time, users and developers are surprising us by finding new uses for Apple Data Detectors. For example, one customer considered how scientific analyses might be started on a Macintosh using Apple Data Detectors to find structures of interest, with subsequent stages of analysis carried out by opening a network connection and passing the detected information to a specialized application running on a Unix workstation. This and other examples emphasize the importance of the scripting layer in Apple Data Detectors and the need to do more than simply launch applications in response to the detection of a data type.

We also see evidence that end users with varying degrees of programming skill are extending actions much as we had expected. One user—a skilled programmer—used the basic detectors and actions that ship with the product as the basis for a new detector/action pair for a personally relevant task. He wanted to look up software bug reports in a database based on the ID numbers of the reports commonly found in other bug reports, email messages, and other program management documents. He distributed this detector/action throughout his work group, and a colleague—a marketing manager with much less programming experience than the original programmer—was able to adapt the action so the detector could run on his laptop computer (where his email and other documents reside) and display the bug reports on his desktop machine (where his program management tools reside). This extension required changes to only a few lines of code in the action, something well within his technical ability. Such end-user tailorability is an excellent example of Apple Data Detectors' evolution. We expect to see more of these innovations—starting with the technology originator (that is, Apple) defining highly general detectors and actions, to detectors appropriate to smaller work groups or communities, to modifications to those detectors to make them support individual users' specific needs. It is important to note that the ease of programming offered by both the detector language and the scripting language used to create new actions (AppleScript) makes this evolution possible. For example, the programmer who wrote the initial bug-tracking action probably could have written his code in C++ or some other production programming language, but it is doubtful that the marketing manager could have made his personal refinements in C++.

Commercialization

Apple Data Detectors started out as a project in the Apple Advanced Technology Group, which designed and developed the initial prototypes. As a research group, ATG had no facilities for shipping products; hence commercialization of the idea was possible only by working with the product development side of Apple. It is worth reflecting on how this happened, as some interesting perspectives on technology transfer follow.

First, Apple Data Detectors offered a crisp statement of value to the product group. The system's capabilities were easy to convey and its value was easy to demonstrate, even with a limited demo system. The engineering required to implement the system was also easy to estimate, partly based on the properties of early demo systems and partly on the system's overall architecture. There was nothing in Apple Data Detectors that would require unrealistically large amounts of memory or processing time, nor would the system impose unrealistic demands on Apple's application developer community. Hence, Apple Data Detectors was the right "grain size" for successful technology transfer—large enough that significant value would extend to Apple customers, small enough that the implications on the operating system were limited and manageable. As a result, the question about Apple Data Detectors was not whether there was a viable product but what form the product would take.

We were fortunate that the technology-transfer question took this form from the beginning, so we were able to start working with the Apple product groups on the technology's product implications early in the technology's development. The transition from technology to product is a long one, and exploring possibilities that ultimately turn out to be blind alleys is inevitable. In our case, the result was something of a dance between research and product groups. We all explored, from our own perspectives, opportunities for the technology; each group revised its understanding and implementation of the technology as these explorations advanced. From the perspective of the research group, we began to see how the product group viewed Apple Data Detectors, and we adapted our development in response, incorporating good ideas from the other side and pushing back when misinterpretations of our work occurred. Meanwhile, all the groups were working from the assumption that some sort of product would emerge. The product group agreed that ATG should have free rein in defining what the technology would be and the product groups could look for opportunities for the technology while not actively investing engineering resources in its development, knowing our work would ultimately become available to them. Therefore we were not in intellectual competition with our own product group. Overall, the process that emerged meant we could avoid the not-invented-here problem that often plagues technology-transfer activities.

The major challenges in our effort to create a commercial product lay in the changes to the system architecture and the user experience Apple Data Detectors would impose on the Macintosh. As the long-term owner of both aspects of Apple's work, the product group needed to ensure the smooth integration of this new technology into the existing system, and it was clear from the beginning that our success in moving Apple Data Detectors into product form depended on our doing this well.

Our early contact with the product group again served us well. During the system's development, we experimented with various user interfaces to its basic technology, an effort that paid off in a number of ways. The different interfaces followed different assumptions about various aspects of the system, such as the interface techniques themselves, the demand the techniques would place on the underlying operating system, and the demand imposed on developers who wanted to adapt Apple Data Detectors to their own uses. It was important for us to understand such trade-offs, so we could advise the product group on the technology's possibilities.

USERS CAN BE IMPRECISE, THROWING THE SYSTEM A BROAD HINT THAT THERE IS SOMETHING OF INTEREST, THEN LET THE SYSTEM USE ITS KNOWLEDGE TO DO THE RIGHT THING.

Agents of Alienation?

Jaron Lanier, a developer of virtual reality technologies, wrote a thought-provoking diatribe against intelligent agents [6], asserting that the very idea of intelligent agents is "both wrong and evil." He argued that while few intelligent software agents have shipped (Bob and the Apple Newton being two of this rare breed), the idea of agents is still harmful, leading to alienation. Lanier finds the idea that the computer has its own intelligence threatening and thinks it undermines human values.

Lanier further argued that agents have to force users to "change yourself to make the agent look smart" and that users thereby "diminish" themselves. But we do not see that users in any way diminish themselves with agents, such as Apple Data Detectors. Users are simply interacting with a program that recognizes data it has been instructed to recognize, taking actions it has been instructed to take. There is less effort needed to make these everyday actions happen, just as when you tell a secretary, "Fax this to Sally right away." You don't have to say what you mean by "right away", "this", and "Sally" or specify Sally's fax number or instruct the secretary to fill out the top sheet of the fax. It's all understood.

We can imagine users of Apple Data Detectors changing their behavior to take advantage of the system's abilities without evil consequences. It is easier to write grammars when the data are regular and structured, and parsing is easier and faster when the data are regular. For example, it may be that in some environments, groups of users establish form-like templates for announcing meetings, so Apple Data Detectors can be used with great speed and accuracy. This activity doesn't compromise or diminish anyone's humanity; it is a natural co-evolution of tools and people. Users themselves change their own behavior if they perceive sufficient value in doing so. There is nothing inherently coercive in agent software.

Our prediction about intelligent software agents is that the best of them will be malleable, programmable tools that empower, rather than diminish, users, giving them control over tasks necessary for everyday life. Collaborative, programmable agents enable users to get more value out of the data they already use for tasks they already do with less effort and at a higher level of goal specification. Such agents will also provide new functions as people discover new uses for them. Apple Data Detectors is designed to support an evolutionary process in which users shape their own tools to the greatest extent possible. Lanier asked an important question: "How does [information technology] affect our definition of what a person is?" Technologies like Apple Data Detectors assume people shape their everyday tools, controlling their own environments through these tools.

Future Directions

Apple Data Detectors is a first step toward extracting semantics from everyday documents without asking users to create documents in new ways. Such an intelligent agent redefines "document" from a stream of characters to a data structure containing specific, known kinds of structures that can play specific, known roles in user interactions. Such an approach can provide a foundation for more powerful analyses beyond our current recognition and parsing technology. Future work could explore the use of more sophisticated kinds of recognition and parsing, including those that rely on finite state technology and linguistically informed context analysis [5], as well as integration with statistical techniques of data analysis, such as relevance-based techniques.

One important future step for research will be to build knowledge into the system about the structures being recognizing and how these structures are related to user goals and tasks. Doing so will provide the basis for more flexible and powerful task support. For instance, if we can attribute some reasonable set of email address semantics to the textual presentation of an email address in a document, the system can use the address as a pointer to the person with that address. We can then carry out system actions intended for people (such as "Place a phone call to this person") on an email address and let the system figure out the person implied by the address. (This can already be done through Apple Data Detectors but requires writing a script, rather than relying on inferencing as suggested here.) Such interaction might require a different user interface from the one used today. One can certainly imagine many different kinds of user interfaces to the basic structure-detection technology underlying the current system.

Along with a growing number of individual grammars in the system, the possibility of ambiguous or inappropriate interpretations of user information also increases, especially given the relatively weak context-free grammars and parsers we are working with. It is easy to imagine a company might choose a syntax for its product order numbers—a three-digit department code followed by a dash followed by a four-digit product code—that would overlap with U.S. telephone number syntax, thus leading Apple Data Detectors to offer both telephone number and part-ordering actions for phone numbers and product codes. We can do little about these overlapping syntaxes without performing a much deeper, semantic interpretation of the text in which the pattern appears, and it is not clear that such processing could be done in the limited time available for Apple Data Detectors' text analysis.

We have also seen finer ambiguities in interpretation that may be resolvable in future versions of the system. For example, given the URL www.apple.com/doit.html, the current Apple Data Detectors finds the intended URL as well as the somewhat surprising pattern of "it.html." This pattern is, however, recognized by the Newsgroup detector as an instance of a Usenet newsgroup, which, like many newsgroups based in Italy, begins with the characters "it.". This interpretation clearly seems incorrect, and time-efficient parsers that avoid such interpretations could likely be constructed. However, we have found that such problems do not detract from the value of Apple Data Detectors because users typically understand why the misinterpretation occurred, and the design of the user interface makes it easy for them to ignore it.

Beyond such enhancements, another goal is to complete a prototype of an end-user programming facility to enable end users to program detectors and actions, opening up the full Apple Data Detectors capability to all users. After all, Apple Data Detectors is meant to empower end users, and our intent is to enable people to play a role in the creation of tools that is as active as their use of them. C

References

1. Barreau, D., and Nardi, B. Finding and reminding: File organization from the desktop. *SIGCHI Bulletin 27*, 3 (July 1995), 39–43.
2. Benyon, D., and Murray, D. Developing adaptive systems to fit individual aptitudes. In *Proceedings of the 1993 International Workshop on Intelligent User Interfaces* (Orlando, Fla., 1993).
3. Cohen, P., Cheyer, A., Wang, M., and Baeg, S. An open agent architecture. In *Proceedings of the AAAI Spring Symposium Series on Software Agents*, O. Etzioni, Ed. (Stanford, Calif., March 1994), American Association for Artificial Intelligence, Menlo Park, Calif., 1994, pp. 1–8.
4. Etzioni, O., and Weld, D. A Softbot-based interface. *Commun. ACM 37*, 7 (July 1994), 72–76.
5. Kennedy, C., and Boguraev, B. Anaphora in a wider context: Tracking discourse referents. In *Proceedings of the 12th European Conference on Artificial Intelligence* (Budapest, Aug. 11–16, 1996), pp. 582–586.
6. Lanier, J. Agents of alienation. *Interactions 2*, 3 (1995), 66–72.
7. Maes, P. Agents that reduce work and information overload. *Commun. ACM 37*, 7 (July 1994), 31–40.
8. Nardi, B. *A Small Matter of Programming: Perspectives on End User Computing*. MIT Press, Cambridge, Mass., 1993.
9. Nardi, B., and O'Day, V. Intelligent agents: What we learned at the library. *Libri 46*, 3 (Sept. 1996), 59–88.
10. Pandit, M., and Kalbag, S. The selection recognition agent: Instant access to relevant information and operations. In *Proceedings of Intelligent User Interfaces '97*, ACM Press, New York, 1997.
11. Rus, D., and Subramanian, D. Multimedia RISSC Informatics. In *Proceedings of the 2nd International Conference on Information and Knowledge Management*. (Washington, D.C., 1993), ACM Press, New York, 1993, pp. 283–294.
12. Shneiderman, B. Looking for the bright side of user interface agents. *Interactions* (Jan. 1995), 13–15.

Bonnie A. Nardi (nardi@research.att.com) is a researcher in AT&T Labs Research in Menlo Park, Calif. She is a former researcher in Apple's Advanced Technology Group.
James R. Miller (jmiller@acm.org) is the former program manager for intelligent systems in Apple's Advanced Technology Group.
David J. Wright (dave.wright@apple.com) is a programmer in Apple's Operating Systems Technologies group, further developing Apple Data Detectors.