Rethinking Autonomy

Richard Alterman (alterman@cs.brandeis.edu) *

Brandeis University

Abstract. This paper explores the assumption of autonomy. Several arguments are presented against the assumption of runtime autonomy as a principle of design for artificial intelligence systems. The arguments vary from being theoretical, to practical, and to analytic. The latter parts of the paper focus on one strategy for building non-autonomous systems (the practice view). One critical theme is that intelligence is not located in the system alone, it emerges from a history of interactions among user, builder, and designer over a given set of data as mediated by the system. A second critical theme is that artificially intelligent systems are ongoing projects that must be continuously adapted and revised using joint person-machine efforts.

Keywords: autonomy, building systems, users, practice, everyday activity, artificial intelligence, cognitive science

* This work was supported in part by ONR (N00014-96-1-0440). Additional funding was provided by NSF (ISI-9634102).

1. Introduction

Part of the ideology of artificial intelligence is that machine intelligence will eventually be *autonomous*; by autonomous it is meant systems that can think, reason or act on their own. Burgeoning developments in "autonomous" agents, softbot and robot, are the most recent example of the pervasiveness of this ideology. But why autonomy? What assumptions are bundled up in the use of this term? What research commitments are being made by framing movements within AI in terms of autonomy? Are these commitments realistic? Is it a good way to design and build a system? What is the alternative? These and related questions are investigated in this paper.

Some potential advantages of an autonomous system are: it compensates for the user's relative lack of sophistication with software, and it maximizes helpfulness while minimizing user work. However, the desire to build an effective autonomous system does not necessarily make it plausible.

The early parts of the paper present arguments against runtime autonomy as a design principle for building artificial intelligence (AI) systems: People are less autonomous in their everyday adaptive behavior than has been previously assumed. Commitment to autonomy is an arbitrary principle of design to assume in building AI systems.

Converting user work into system intelligence will significantly increase the performance and flexibility of the non-autonomous system over its autonomous counterpart. An AI system mediates a set of practices between people with different roles and responsibilities, skills, and goals — an important member of this group is the user.

The latter parts of the paper explore the consequences of one strategy for building non-autonomous AI systems (the *practice view*). The underlying hypothesis of the discussion is that the pragmatics of decision making in using a system and the emerging practice of the user are inseparable features of the intelligence of any AI system. In contrast to two other frameworks for non-autonomous AI — *collaboration* and *distributed cognition* — that assume a model of user interaction for single episodes of problem solving, the practice view focuses on the emergence of intelligence from multiple episodes of user system interaction. A consequence of the practice view is that the development of a system is a continuous and ongoing project. During the initial development of the system the designer and the system builder are in charge of developing the system. After deployment the system must continue to develop, a project jointly undertaken by user and system.

2. Designed Environment

For a moment, imagine that it was possible to add to every man-made object (and its parts) bar codes that provide information about the usage and function of that object. An important research problem is to devise a useful set of bar codes. For example what would be a reasonable set of codes to install in your house to be used by a "house cleaning robot"? How many codes would need to be installed? Where? Here are some the characteristics of what would make a code useful:

1. The codes communicate information for a large variety of tasks.
2. The information contained in the codes are useful for an extended period of time. Two complicating factors are:
   a) The robots vary and they will be evolving.
   b) New, and changing, task requirements will need to use existing codes.

This problem is the design problem (Norman, 1988). One might say that if mechanical agents were provided with this kind of information, the agents themselves would not be very intelligent at all. But the opposite is just the case. These are exactly the conditions of our everyday activities. As I look around me, each of the objects in my office (the telephone, the keyboard, buttons and switches, the mouse, et cetera)

are designed to communicate their usage. If the world was not encoded by some other in a manner that would allow me to "read the world", the range and effectiveness of my activity would be drastically reduced.

This view of everyday activity is an interactionist account, but it requires more than a perception/action loop. The individual must be able to "read" information provided at the scene of the activity that is expressly put there to aid the individual in adapting and learning.

3. Everyday Activity

FLOABN (**F**or **L**ack **O**f **A** **B**etter **N**ame) is a cognitive model of an individual acquiring skill in the usage of household and office devices (Alterman et al, 1998, pp. 53-105). The underlying behavior of FLOABN is based on *adaptive planning*, re-using and refitting old procedures, rather than planning from first principles (Alterman,1988, pp. 393-421). A critical feature of the FLOABN model is that adaptation is mediated by instructional help provided by another individual. Figure 1 depicts FLOABN using a photocopier.

The processes by which an individual (as modeled by FLOABN) adapts to both the evolution and variance in interface design are guided by the direct (and indirect) instructional communication between de-

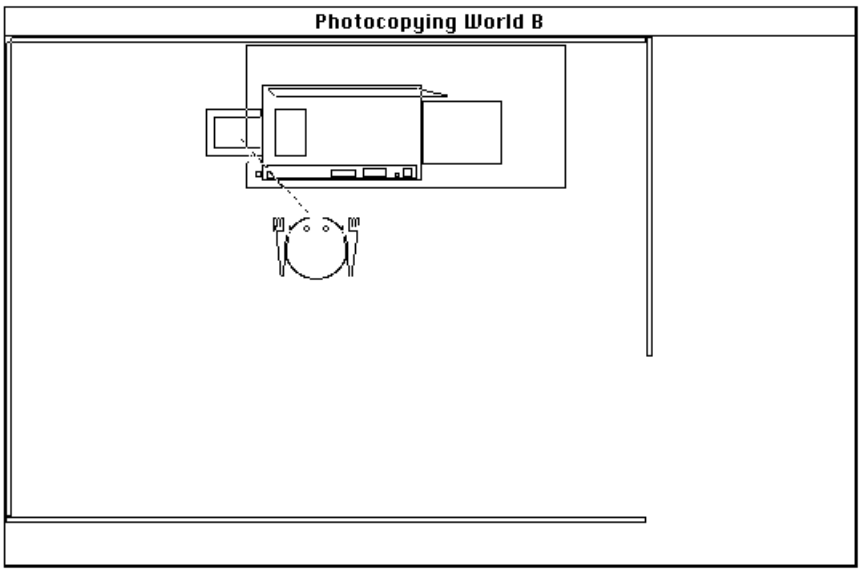

Figure 1. FLOABN Looking for the Output Tray in the Copier World

signers and users. From the designer's point of view, building the interface is a problem of communication. Resources for communicating to users how to use a device include written or spoken instruction, the presentation of the device, labels, diagrams, and various iconographs. As action unfolds, the user avails him/herself of the instructional information available at the scene of the activity. An icon of a credit card conveys the "mode of payment," the design of the telephone on an airplane head piece affords listening, and the instructions tell the actor to "insert the credit card" before lifting the handset. Because there is no a priori analysis that can exactly predict the sequence of actions to perform, the individual must base his/her activity on the signs and instructional information provided to guide the individual's activity.

Examples of everyday tasks other than device usage that rely on help from others during activity are ubiquitous.

Opening a door is an example of a simple physical act that occurs in the everyday world. Norman (1988: pp. 3-4) provides an anecdotal example of how such a simple task as entering through a doorway becomes complicated without the right "signs". An individual is entering a building by passing through a series of doorways. Halfway through the doorways the individual becomes distracted, disoriented, and because the hinges of the doorway are not visible, becomes confused about the correct place to push on the door in order to open it. In other words, without help from another, the "simple" thinking required to open a door becomes much more complicated.

Whether it be navigating in one's home town or as a visitor in another town, navigation is guided by information provided by another, and tied to the 'landmarks' already known by the individual. The ways in which the individual actor can get help from another in navigating include:

1. asking someone
2. using a map (street or building)
3. using a sign (street names and numbers; office names and numbers)
4. default reasoning (about layout of cities, buildings, hallways)

All three methods of guiding navigation require access to others, either directly, or via an artifact or sign. In no case does the navigating individual act and think alone, separated from the world.

High-level mental activities like reading, conversation, and theorem proving all have critical aspects that are non-autonomous. A reader has goals and plans (Carpenter & Alterman, 1994, pp. 62-67). As the reading activity unfolds, the reading plan is adapted using information explicitly provided in the external world for this very purpose; the structure of the text and the formatting of the text (e.g., boldfacing, enumerated lists, and sections headings) are explicitly added to the text in order to guide the reader. That is, the reader uses the organization of the text — as communicated by the author to the reader through the structure of the text — to reason about the text. In Clark's book on using language (Clark 1996), he provides evidence that in conversation the creation of meaning on the part of the speaker is a joint action requiring efforts from both the speaker and the hearer. A protocol from of Anderson's work (Anderson, 1982, pp. 369-406) on cognitive skill acquisition, as it applies to the generation of Euclidean geometry proofs, suggests that students are reasoning about the everyday aspects of this task (p. 382):

> Right off the top of my head I am going to take a guess at what I am supposed to do: $\angle DCK \cong \angle ABK$. There is only one of two and the side-angle-side postulates is what *they* are getting to.[1]

Who is the "they"? Presumably it is the authors of the text. This as an indication that the students are assuming that the textbook has been designed so as to guide the student's problem-solving activity.

In each of these everyday activities, regardless of whether they reflect high or low levels levels of thought, the individual actor is not adapting in isolation. In each case, another actor provides information relevant to the individual's performance of the task — the individual's adaptive behavior is not autonomous.

There is a widespread belief in AI (and Cognitive Science) that the dynamics of the world make continuous adaptation and learning a critical feature of intelligence. If the availability of information provided by another is a necessary condition of success in accomplishing a tasks in the everyday world, then the idea that people are thinking and acting in a "purely autonomously manner" is at best problematic. Given that the smartest creature around does not adapt and learn on her own, what exactly is the motivation for building AI systems that at runtime must work autonomously?

4. Get Ready and Run

Any implemented AI system cycles through two stages of development:

1. getting ready
2. running

In the first stage, the system is prepared for some task and task environment; in the second stage the system "runs" according to its design. The decision of whether a given stage is autonomous or not is a separate one. The tradition in AI has been that the "getting ready" stage is non-autonomous and the "running" stage is autonomous; this applies equally to the construction of insects, parsing natural language text, planning, or whatever.

From the perspective of the history of the development of a given AI system, it is the task of the system builder to *get the system ready* for a particular task environment; this preparation phase does not occur autonomously. Much of the learning about the task environment — deciding how and what to represent about it — actively involves the efforts of the system developer. During this non-autonomous stage, the system builder simultaneously prepares the system for running and designs the task environment in which the system will run: a set of

data is identified, decisions are made about how to represent the data, information to exclude or include; and similarity and heuristic functions are devised. The developer must also select which artificial intelligence techniques to use. The ordering of these tasks is undetermined; sometimes the data comes first, other times it is the AI technique that is selected first; in most cases, the developer moves back-and-forth remaking each of the decisions. It is significant that the getting ready stage requires learning about the task and the task environment and that it is not done automatically or autonomously. After the non-autonomous stage is complete, hooking up the data to the technique, the system is deployed and runs "autonomously". *Of course, if the system does something funny, it will be necessary to cycle again through the stages of "non-autonomous" and "autonomous" learning.*

Improvement in the performance of a system occurs at both stages of development. During the "get ready" stage, the developer builds the theory of the system based on what he/she learns about the task and the task environment; during the "run" stage, lessons are learned about the application of the system which subsequently result in changes to the system during the next period of "get ready". When the system is running, it is autonomous — it runs "on its own" in the task environment. If you looked at the development of a given AI system, it is likely that it has switched back and forth between the processes of

"getting ready" and "running" throughout its development. *Usually a system is defined as autonomous if on the days that it is "demo-ed" it runs without human intervention or aid.*

If we look at the development of a given artificial intelligence system, it is not clear how much of the credit for the success of the system goes to the non-autonomous stage and how much should go to the autonomous stage. Anybody who has ever taken an introductory AI course has hands-on knowledge as to how dependent the success of any AI technique is on the efforts of the programmer. Take a simple heuristic search program. Critical to the success of the endeavor is that the programmer must educate him/herself about the task in order to build a useful representation scheme and an effective heuristic to plug into the selected "search technique".

Consider a system which exhibits a certain level of intelligence. Compare the contribution of the development of the system (several cycles of get ready and run) to the contribution of the automatic learning that occurs during one stage of "running". It strikes me that the "learning" is a secondary effect on the development of the system. *If the system is intelligent, credit largely goes to how it was developed which is a joint person-machine practice.*

From a system building perspective, the adoptation of "autonomy" as a principle of engineer is arbitrary. A critical part of building the

system is learning about the task and the task environment. What exactly are the advantages of making the system run autonomously? What are the possible advantages of making the user a part of the process making the system intelligent?

5. The Work and Practice of the User

5.1. The Work of the User

There is payoff for the AI engineer in building a system that redistributes the work of making the system "smart" between developers and users. By making the "running system" non-autonomous, the development of the system becomes continuous, drawing on the separate sources of knowledge that are available during the "get ready" and "run" stages of the system.

During the running stage the non-autonomous system can potentially get advice on adapting and learning as-it-goes-along, just as is the case for humans in their everyday task environments. Consider as an example the development of a symbolic-based artificial intelligence system. The developer is, in general, concerned with questions of structure: representation schemes need to be devised that provide an ontology

and syntax for the reasoning done by the system. Both similarity and heuristic evaluation are functions over the representation. Control is a function of similarity and heuristic evaluation of state. Each of the decisions (and the intelligence it embodies) results from the practice of the AI system builder. Additional intelligence results from the user's work. Given a theory of how the data is to be used — as it is embodied in the system — the user can make pragmatic judgments as to how to represent a particular piece of data. The user can make judgments about the similarity of two particular items or evaluate the "goodness" of a particular state. The user can make control decisions.

Other advantages of the non-autonomous scheme derive from the differences in expertise between developers and users. The developer's area of expertise is the system. The user's area of expertise is the task environment; for domains with experts, the user's actions reflect expertise about the domain. The system developer may not have these kinds of knowledge of the domain either because she is building a shell (and does not know the eventual application) or she knows the application but lacks expertise in the domain. Even where the developer has some knowledge of the domain, there will be regularities in the system's application to a given domain that are best learned after the system is deployed.

5.2. The Practice of the User

Over an extended period of time, a practice emerges from user's application of the system to a particular set of tasks. The set of experiences that are generated in the course of the user's continued work over many sessions of application are a source of information that can be used to adjust the system's behavior so as to better fit the user to the system and the system to its application to the user's data sets. Over multiple episodes of problem solving with the system, the practice that develops is a significant part of the emergence of intelligence in a person machine interaction.

After deployment the system can continue to develop by including mechanisms in the implementation of the system that allow the system to *learn* about the user's application of it to a given domain. Information is accumulated about the user's practice that can be subsequently used as a basis for making practical inferences about the application of the system in a particular task environment. The resulting adaptations fit the design of the system to the task environment of the application by customizing it to regularities in either the data or usage of the data or the user's interaction with the interface.

Adding components to the system that reason about the developing user practice have two general advantages over a designed intelligence only scheme.

1. Learning the specifics of the task environment.

2. Mitigating effects of imperfections in the design process.

In many ways, the initial design of a system is like a plan for how a system is to achieve certain ends in a given task environment. Details of the application and the task environment can lead to improvements in system performance that are unachievable in the general case. However, during the development process much of this kind of information may not be available. By including runtime learning mechanisms in the design of the system, some of this kind of knowledge, as it is reflected in the practice of the user, can be gathered after the system is deployed.

The other potential benefit derives from the imperfect nature of the process of designing and developing an AI system. In general, it may not be possible to determine the best design for a given application. At any given point in time, there may be several candidate techniques that could be used for the same task in the application. It is very difficult to select amongst those candidates, since each of them has conditions which may, or may not, exactly match the conditions of the application. Moreover, all of the tinkering that occurs during the

development can effect in unpredictable ways the performance of the designed system. Even if there was a "best technique" for a given task in an application, the developer may not know what that technique is, or may be more familiar with, or invested money in, another technology that does almost as good but requires less time to implement. Some of the effort in finding and implementing the "best design" of the tool function in the initial implementation may be better spent in adding runtime learning components to the system.

In some cases, very simple and obvious strategies for learning from the user's practice may significantly improve system performance. Alterman & Griffin (1996) give evidence that for some tasks (in this case building a CBR system), learning and adaptation are more easily accomplished by substituting a *runtime learner* for system builder effort. User marks of positive or negative evaluation were included in the run time data used as input by a learner. In this study, the preexisting data that was used to generate the initial set of cases had some problems, mostly deriving from the fact that a uniform vocabulary was not established for representing the cases. Because of these conditions, the clustering and retrieval were problematic; i.e., recall and precision numbers for the initially deployed system were low. *Rather than spending enormous amounts of time rebuilding and reclassifying the data based on the system builder's evaluation of system performance, the system*

adapted to the user's evaluation of the relevance of retrieved cases. Retrieval techniques that reasoned using the user's input were tested, and a significant increase in system performance was achieved. In a follow up study, Alterman & Griffin (1997) constructed an alternate design for the help desk retrieval system. The second design was reputed to be the significantly better than the other (without adaptation) for a text retrieval problem; the results confirmed this. The adaptive mechanism developed in the first study was added to the second system. Experimental results showed that either adapted system was better than either non-adapted design, and that the results were better regardless of which design you began with. Moreover, the performance of the two adapted systems began to converge.

6. System as Mediating Artifact

An artifact is defined as "any object made by human work" (Websters (1970), pp. 84). A mediating artifact is an artifact that mediates the interaction between a subject and an object. A simple example of a mediating artifact is a shovel, which in the course of digging mediates the interaction between the digger (the subject) and the ground (the object). Frequently the relations between subject, object, and mediat-

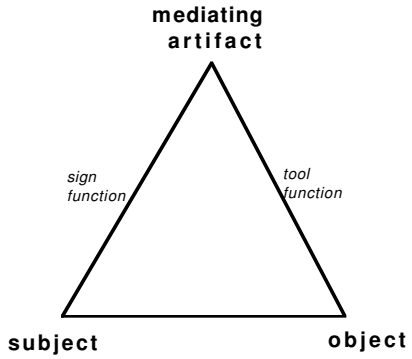

Figure 2. Mediation Triangle

ing artifact are depicted as triangular (see Figure 2) and referred to as a *mediation triangle* (Cole & Engestrom,1993, p. 5] . A mediating artifact functions both as a *sign* and and as a *tool* [Vygotsky, (1978), pp. 54-55]. The tool function is "to serve as the conductor of human influence on the object of activity." The sign function is oriented towards the subject, and over time it becomes internalized as a part of the mental function of the individual. The work of Scribner (1984) on arithmetic in the milk processing plant gives a good example of the sign and tool functions of a mediating artifact. In the case of Scribner work, the mediating artifact is the milk crate, which holds sixteen quarts of milk. As a tool it is used for carrying quarts of milk. As a sign, it played an important functional role for the mental aspects of the pricing of delivery tickets. Rather than calculating a given order by multiplying the unit price for, say, 32 quarts of milk, the price was calculated as twice the cost of the standard 16 quart case.

By definition any AI system is a-kind-of artifact. The usage of a system is a mediated behavior; the system is the medium by which the user manages to perform a particular task on a given set of data. The sign function of the system provides the basis of communication with the user, and the tool function, a set of methods for manipulating the data.

The mediation triangle applies to both phases of the life cycle of an artificial intelligence system. During the "get ready" stage, the system mediates the designer's and engineer's interactions with the data, and during the "run" stage, the user's interaction with the data.[2] When a given system succeeds, credit should be given to all three of the vertices of the mediation triangle. The efforts of the builder and user of the system, the technique that the system embodies, and the design of the task environment all deserve credit for the success, or failure, of the system's adaptive behavior. From the perspective of the mediation triangle, leaving the user and the usage of a given invocation of a system out of the development process is an arbitrary engineering decision.

7. Joint Runtime Learning

Making the system run non-autonomously for single episodes of problem solving can improve the system's performance on the current problem. The set of experiences that are generated in the course of the user's continued work over many sessions of application are a source of information that can be used to adjust the system's behavior so as to better fit the user to the system and the system to its application to the user's data sets. Over multiple episodes of problem solving with the system, the practice that develops is an significant part of the emergence of intelligence in a person machine interaction.

A range of runtime learners can be differentiated by commitment to system autonomy (see Figure 3). One end point is the system that runs autonomously and learns from its autonomous behavior. The other end point is where the user is responsible for all runtime adjustments to the system. In between are two approaches to building runtime learners that include input from the user. I will refer to this middle group of runtime learns as *joint runtime learners* because it involves both user and system effort. In a joint runtime learner the user learns about applying the system/tool to a task environment, and the system learns about the user's emerging practice. Joint runtime learners come in two types, differentiated by the kind of input they take from the user. Joint

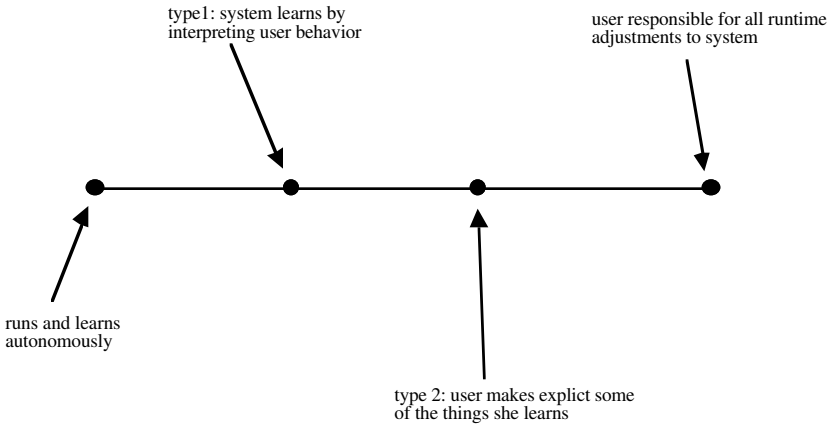

Figure 3. Runtime learners.

runtime learners include either implicitly (type 1) or explicitly (type 2) the lessons learned by the user about the system's application to the data.

Consider a learning problem like the acquisition of macro operators (macrops). As initially presented by Fikes, Hart, and Nilsson (1972), the macrops learner is an runtime learner that is piggybacked on top of an autonomous system. However, it is possible to create type 1 or 2 input data for the macrops learner. Type 1 could include problems partially, or even entirely, solved by a user. Type 2 could include macrops that were created by the user or judged by the user to be significant. For task environments where expertise is available, type 1 and 2 input data directly exploit that expertise. But even when experts are not available, the input data includes the user's pragmatic evaluations of the system's application to a given set of data.

For type 1 data, it is the responsibility of the system to interprete the user's behavior; it has to construct an explanation of what the user is doing. In some cases, this may be as simple as recording each of the user decisions but this leaves to the learner the task of evaluating which of the user decisions were propitious and which were not. If a causal interpretation of the user behavior becomes necessary, the system builder is burdened with the task of building into the learning system a theory that is robust enough to account for much of the user's behavior. This sort of high level interpretation is notoriously difficult for any system to do.

In general, it is both an advantage and a disadvantage that type 1 runtime data requires no direct communication between the user and the learner. The advantage is that no language must be established for communication, and the disadvantage is that the user has no control over how the learner interprets his/her behavior. One of the difficulties of asking users to provide type 2 information is that, despite eventual improvements in system performance, users do not want to be bothered with doing work that has no direct benefit. A second difficulty is that a language process has to be established by which communication between person and machine can occur in generation of type 2 data.

In some cases, very simple and obvious strategies for generating type 2 runtime data may significantly improve system performance. The

Alterman & Griffin (1996) work described in Section 5 gives evidence that for some tasks (in this case building a CBR system), learning and adaptation are more easily accomplished by substituting a learner that uses type 2 data for system building effort. Another example of a language process that has the right mix of capabilities and could be used to generate type 2 data is analogous to the process of "reading and marking text" which is commonly used as a method for highlighting or evaluating certain information in a text. The basic idea is that the system produces behavior, the user reads that behavior, and marks it. Input to the learner includes both the system produced data and the secondary structures provided by the markings of the user. The advantage of user marks as a language for communication are twofold. The semantics of the marks is straightforward to interpret, and user marks allow the user to enforce some control decisions on the learner.

Consider an online manual system as an example of how this process might fit the requirement of knitting together the task with the generation of type 2 data. One important characteristic of the user's reading of the online manual will be how it is structured and formatted; the structure and format is information provided by the author of the text to provide guidance in the interpretation of the text. One of the advantages of allowing the user to mark the text is that it allows the user to create structure as she/he goes along. Each time the online

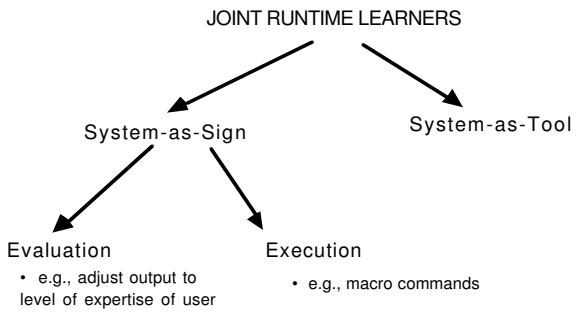

Figure 4. A Taxonomy of Joint Runtime Learning Systems

manual is used, the user is adding temporary structure to the data which did not exist previously; when relevant information is found it can be marked and saved off in a scratch pad for later consultation. The user's marking behavior potentially serves two important functions. It improves the user's performance, and it creates type 2 data, candidate structures for organizing the online manual that can be used as input to the learning system. In the long run, if the system collects this information, it can be used to add dynamically structure to an online manual.

7.1. Two Functions

Joint runtime learners can also be divided into two kinds that are differentiated by the system functions that they effect (see Figure 4). One kind are those that adapt the designed system by changing the sign aspect of the system; this kind of system is the norm in the

HCI community and is generally referred to as an *adaptive system*. Their role has been to reduce human effort in using the interface by customizing the interface to the user. A second kind of joint runtime learner are those develop the designed system by changing the tool aspect of the system; these sorts of techniques originate more in the artificial intelligence community. There role is to improve performance of the system in its application to the task environment.

Systems that adapt the sign function of the system can be further divided in terms of Norman's (1988) distinction between the *execution* and the *evaluation* aspects of communicating with a system. The execution function of the interface refers to the set of actions that the user can take. The evaluation function of the interface refers to the representations provided by the system that can be directly perceived and that are directly interpretable in terms of the intentions and expectations of the user. An example of a system that adapts the execution function of the system are those that allow for the creation of macro operators. Examples of systems that adapt the evaluative function of the interface include systems that adapt dialogues, or the presentation of information to the individual characteristics of the user.

Where sign adaptation helps the user to better express her intentions and read the state of the system, tool adaptation helps the system to better carry out the user's intentions in the task environment. Where

sign adaptations reduce the user efforts in using the interface, tool adaptations improve the systems performance of the task. Many of the more recent applications for joint runtime learners that develop the tool aspect of the system have to do with retrieving information from online sources. In general, they improve the tool performance by recording the judgments of users about items retrieved, inspected, or searched. Examples of retrieval systems with components that improve system performance as a function of use are systems that do collaborative filtering, track user preferences, suggest hyperlinks to aid user search, and recognize questions that have been previously asked. Other domains where joint runtime learners have been used to extend the tool performance of the system are knowledge acquisition and mixed initiative planning.

8. Other Models of Non-Autonomy

The model of a non-autonomous system presented in this paper is framed in the terms of the emerging user practice. There are other models (and frameworks) for non-autonomous systems.

Grosz (1996; Grosz and Kraus, 1996) defines collaboration as working "jointly with" somebody else. She differentiates collaboration, jointly

working with a user, from a master-slave interaction. Participants in a collaboration must share information; they have different capabilities; they share an expectation that each will attempt to overcome any difficulties that arise in the course of action. Collaborating individuals develop *shared plans* (Grosz & Sidner, 1990, pp. 417-444), to which each of the collaborators are committed. Agreeing on a shared plan, dealing with resource conflicts the arise between collaborators, overcoming unanticipated problems, group decision making and negotiation, are all activities that are part of the collaborative process. They all also require communication.

There are three different kinds of collaboration that Grosz talks about: person to person, software agent to software agent, and human to computer. The model of collaboration that Grosz and her colleagues provide plays out differently for each of these three kinds of collaboration. Person to person communication is quite effective and a robust version of collaboration is possible. A collaboration between software agents communication will have different characteristics, but in a closed world may be possible. Both person to person and software agent to software agent are collaborations among entities with similar capabilities, but a human computer interaction is an unequal partnership (*symmetric versus asymmetric abilities:* Terveen, 1995, p.73).

Notice a collaboration-based model of human computer interaction commits AI to a framework where the human and the computer are two separate entities. Models originating in the HCI community, frame the interaction between human and computer in the terms of an extension metaphor, the computer is a tool that extends the capabilities of the user.

Norman (1991, p. 17) defines a *cognitive artifact* as follows: "A cognitive artifact is an artificial device designed to maintain, display, or operate upon information in order to serve a representational function." In other words, the cognitive artifact is a mediating artifact that mediates the interaction between a subject (the user) and an object (a representation of information). What is unusual about a cognitive artifact that is an AI system, is that the artifact is capable of heuristically manipulating the information; these heuristics are implemented by statistical, logical and/or syntactic processes.

A critical difference between framing human computer interaction as collaboration or as tool usage is reflected in the requirements of each model for communication between the user and the system. For the collaboration framework the human and the computer are two separate entities. The collaboration model of human computer interaction maximizes the potential input from the user, but it carries heavy communication costs. Where people can manage these, and program-

mers can (perhaps) control them for software agents by decree, getting asymmetric entities (the human and the computer) to manage it is somewhat more difficult (e.g., chapter 6 and 7 of Suchman, 1987).

For the tool model of interaction, the human and the computer are not separate entities, rather the computer and the human jointly define a system (Norman, ibid), and consequently, communication is transformed into *distributed cognition* [Hutchins (1995), pp. 265-288]. One example of capabilities that can be distributed between the user and the program is memory. In Hutchins work (1995, ibid) on the airplane pilot and the cockpit, tasks requiring complex and costly internal memory operations are distributed between the pilot and the instrumentation of the cockpit: significant amounts of what needs to be remembered is located in representations (e.g., the "speed bugs" on the airspeed indicator) available in the task environment. Other examples of capabilities that can be distributed between the user and the program are executive functions (e.g., control of a system), and higher order knowledge (knowledge of standard procedures to follow in the task domain) (Perkins, 1993, pp. 88-110.

A critical difference between the practice-based model presented in this paper and the collaborative and tool-based models, is that it ties non-autonomy to joint runtime learners. A joint runtime learner

converts the user's developing practice into features of the emerging person machine intelligence.

Where both the collaborative and the distributed model of interaction view non-autonomy in the terms of a single episode, the practice-based framework grounds non-autonomous systems in the requirements of multiple related episodes of activity. User practice can emerge from either one of the other two frameworks. Where a shared plan between user and system is developed for a specific task, a shared practice emerges from the completion of several tasks in the same context. The characteristics of distribution between user and system can change over time as a result of continued use of the system over several episodes of interaction.

Because practice is tied to context [Lave (1988)] and particular bodies of knowledge [Scribner & Cole (1981), pp. 236-238], the system is in a position to reason practically on the specifics how the user applies the system to a set of tasks. By building components into the AI system that reason non-autonomously about the user's emerging practice, the system and user can begin to substitute in practical inferences, given the peculiarities of the user's tasks, for abstract, all encompassing, inferencing schemes. Over time, as the user begins to adjust to regularities in the application of the system to the task environment, a user practice

will emerge. Representations of the user's practice are the kinds of things that the AI system can adapt to (learn from).

9. Concluding Remarks

In the everyday world, the individual works in designed environments which provide much guidance for adapting and learning new behaviors. Success in the accomplishment of a given activity is in part attributable to the history of such activities within the community, and in part to the success of the task environment, and the instructional materials it provides, in aiding the individual in adjusting his/her behavior. Just as is the case for humans in their everyday task environments, once a user becomes a part of the process that makes the artifact/system "run" (and adapt) effectively, the system potentially can get advice on adapting and learning as-it-goes-along.

In general, the task environment for a given system is uncertain, dynamic, and in some way non-representable, with relevant information only available at runtime. Success for an AI system depends on drawing practical inferences from the practice of the user in applying the system to a set of problems in a particular task environment. Once a user becomes a part of the running system, intelligence is no longer located

in the system alone, rather it emerges from the history of interactions between system and user.

The generation of intelligent behavior after deployment requires a model of behavior that depends on the continued development through joint person-machine learning throughout the lifetime of the system — a practice-based approach. In the era of web-based AI research, a practice-based approach to non-autonomous AI systems has special significance. Research on the web has emphasized issues in making accessible greater amounts of information to a larger population of individuals. But from the perspective of user's practice, the development of the web offers unparalleled opportunities for AI to have access to huge amounts of user work. If the field can develop technology for converting one percent of user work and practice into system intelligence, the level of intelligence exhibited by AI systems will significantly increase.

In a plenary address at the 1997 AAAI conference James Allen argued that the AI field had moved out of its infancy and had begun to mature as a result of methodological changes in the field. Another earmark of maturity is the ability to adjust the idealizations of youth to the realities of everyday life. As an infant science, AI idealized artificial intelligence as being autonomous. It is time to rethink the assumption of autonomy. The intelligence of a system is indivisible from the people who use it, built it, and designed the task environment in which it runs.

References

Richard Alterman. Adaptive planning. *Cognitive Science*, 12:393–421, 1988.

Richard Alterman and Daniel Griffin. Improving case retrieval by remembering questions. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence*, pages 678–683, 1996.

Richard Alterman and Daniel Griffin. Improvement as a function of use. Technical Report TR-CS-95-182, Brandeis University, 1997.

Richard Alterman, Roland Zito-Wolf, and Tamitha Carpenter. Pragmatic action. *Cognitive Science*, 22(1):53–105, 1998.

John R. Anderson. Acquisition of cognitive skill. *Psychological Review*, 89:369–406, 1982.

Tamitha Carpenter and Richard Alterman. A reading agent. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*. Massachusetts Institute of Technology, 1994.

Michael Cole and Yrjo Engestrom. A cultural historic approach to distributed cognition. In Gavriel Salomon, editor, *Distributed Cognitions*, pages 1–46. Cambridge University Press, 1993.

Richard E. Fikes, Peter E. Hart, and Nils J. Nilsson. Learning and executing generalized robot plans. *Artificial Intelligence*, 3:251–288, 1972.

Barbara Grosz. Collaborative systems. *Artificial Intelligence Magazine*, 17:67–85, 1996.

Barbara Grosz and Sarit Kraus. Collaborative plans for complex group action. *Artificial Intelligence*, 86:269–357, 1996.

Barbara Grosz and Candace Sidner. Plans for discourse. In Phillip R. Cohen, Jerry Morgan, and Martha E. Pollack, editors, *Intentions in Communication*, pages 417–444. Bradford Books, Cambridge, MA, 1990.

David B. Guralnik, editor. *Webster's New World Dictionary of the American Language, Second College Edition*. World Publishing Company, New York, 1970.

Edward Hutchins. How a cockpit remembers its speed. *Cognitive Science*, 19:265–288, 1995.

Jean Lave. *Cognition in Practice*. Cambridge University Press, Cambridge, 1988.

Donald A. Norman. *The Psychology of Everyday Things*. Basic Books, 1988.

Donald A. Norman. Cognitive artifacts. In John M. Carroll, editor, *Designing Interaction*, pages 17–38. Cambridge University Press, 1991.

D. N. Perkins. Person-plus: a distributed view of thinking and learning. In Gavriel Salomon, editor, *Distributed Cognitions*, pages 88–110. Cambridge University Press, 1993.

Sylvia Scribner. Studying working intelligence. In Barbara Rogoff and Jean Lave, editors, *Everyday Cognition*, pages 9–40. Harvard University Press, Cambridge, MA, 1984.

Sylvia Scribner and Michael Cole. *The Psychology of Literacy*. Harvard University Press, Cambridge, MA, 1981.

Lucy A. Suchman. *Plans and Situated Actions*. Cambridge University Press, Cambridge, 1987.

Loren G. Terveen. Overview of human-computer collaboration. *Knowledge-Based Systems*, 8(2-3):67–81, 1995.

Address for Offprints: Richard Alterman, Computer Science Department, Center for Complex Systems, Brandeis University, Waltham, MA 02254.