The Trust Issue in Task Automation

Robert J. Hall
AT&T Bell Laboratories
600 Mountain Ave.
Murray Hill, NJ 07974
hall@research.att.com

Abstract

End-user programming, programming by demonstration, machine learning, AI planning, and knowledge sharing are among the techniques that allow end-users to automate idiosyncratic, mundane tasks. However, a critical issue that has been left virtually unaddressed by the research and commercial communities is why should the user *trust* the acquired task knowledge? That is, how can the user gain confidence that it will suffice to automate the task in a way consistent with the user's goals and priorities, degrading gracefully in the face of failure? This paper first surveys the trust problem and how it arises for each of several acquisition techniques, surveys and compares existing approaches to solving the trust problem. Finally, it briefly overviews the author's novel *incremental validation* approach to trusting behaviors. (A separate paper (Hall 1995c) discusses this approach in detail.)

Introduction: The Assistant Interface Metaphor

Secretaries in traditional offices exist to make a business person's time more cost effective: relatively low-paid, they handle routine tasks flexibly and reliably on behalf of a more highly-paid person, shielding him/her from the low-level execution details. As more and more of a user's business and personal activities are carried out in networked information environments, the potential exists to delegate tasks to (even cheaper) computers instead of to other humans. Most commercial tools, however, while providing new capabilities, do not shield their users from detail; on the contrary, users must often deal with tens of software packages, each of which may have hundreds of functional features. Moreover, whereas a human assistant knows how to use multiple tools to achieve a goal, integrating their work products, today's computer user must deal with multiple application programs "by hand," saving files, cutting and pasting, transferring files among machines, etc. This cognitive complexity and need for manual intervention threaten to negate the efficiency gains sought from increased computerization.

To combat this, we would like computational environments to have *assistant-like interfaces*: interfaces that shield the user from low-level complexity while allowing the automation of mundane, idiosyncratic tasks. To be an acceptable replacement for a human assistant, however, I believe the following properties are critical to the assistant metaphor:

- *Autonomy:* The assistant must be able to carry out routine tasks with little or no intervention, shielding the user from low-level task details. An assistant-like interface is not intended to replace commercial software packages; humans will still use them to compose documents, read mail, play games, etc. Instead, it will automate the routine details that fall in the cracks between them.
- *Trustworthiness:* The user must be able to trust that the assistant will behave in accord with the user's goals and priorities, degrading gracefully when a task cannot be completed by (potentially) both informing the user and aborting the task in the safest and cheapest way.
- *Extensibility:* The user must be able to delegate new, idiosyncratic tasks, with the assistant assuming some of the burden of finding out how to perform them. This *behavior acquisition* capability is critical, because having multiple assistants, each with its own (possibly conflicting) capabilities, would be of questionable value in reducing the cognitive load on the user; and yet each individual user will have idiosyncratic needs that are not fully met by some feature of a commercial software package. Of necessity, commercial packages must appeal to wide audiences to be viable, so they cannot spend resources developing features that are useful only to a few customers. And even if some package somewhere performs the task, it may be unavailable, incompatible, or too expensive to suit the user's needs.

While much research has been done on the problems of task automation (Wayner 1994; Wayner 1995; Sun 1996) and extensibility (Smith et al 1994; Eisenberg & Fischer 1994; Cypher 1991; Mitchell et al 1994; Maes 1994; Hall 1995a), relatively little attention has been devoted to techniques for increasing the trusta-

bility of an assistant. And yet I believe that making an automated assistant more trustable will have a large impact on the structure of the computer-human interaction, because it cannot be "added on" as an invisible, back-end module. Human bosses, after all, do not trust their human secretaries because of written test results. Instead, trust is built incrementally as the boss observes not only the quality of the secretary's results, but also the way in which the secretary works, incrementally tuning the secretary's rules for handling tasks. Moreover, as new tasks are delegated to the secretary, there is often an "apprenticeship period" during which the boss more closely supervises the secretary until s/he comes to trust that the secretary has learned to perform the task in accord with desired goals and priorities. Thus, it seems likely that an assistant-like interface will have to allow for trust building activities and for varying levels of supervision.

The Trust Problem

Assuming one is able to delegate a task to an automated assistant, why should one trust it to carry out the task correctly, or at least to perform as appropriately as possible in the event of failure? This question is critical to user acceptance of an assistant-like interface when significant assets are put at risk, such as money, privacy, or personal reputation (Norman 1994; Riecken et al 1995; Shneiderman 1991). Even worse than a lack of user trust would be to have users that are *too* trusting, as this could cause them significant harm.

Assistant behaviors are harder to trust than commercial software packages, because the latter (usually) have a published specification and a large pool of users who report problems (to the vendor as well as the press). Assistant behaviors, by contrast, are typically idiosyncratic and unsupported.

Consider what can happen when one delegates the purchase (by phone, say) of two concert tickets; the assistant might

- succeed in a timely fashion.
- buy one ticket (e.g., if only one were available).
- buy two less desirable tickets (e.g., second balcony).
- be tricked into paying too much for less desirable seats.
- call the wrong number and give money (e.g., credit card information) to a thief.
- fail silently.
- fail, but mistakenly tell the user it succeeded.

Note that several of these (such as settling for two less desirable tickets when the more desirable ones are sold out) are not obviously erroneous behaviors, but may be acceptable to some users and not to others.

There is a great deal of work on task automation via agent scripting languages (such as Telescript (Wayner 1994), Safe-TCL (Wayner 1995), and Java (Sun 1995)), but these efforts do not directly address the trust problem. While all have mechanisms for limiting agent resource usage, they are silent on the issue of how to determine in advance whether the agent will perform *as the user desires* within its resource bounds. In the ticket-buying example, they can protect against an agent paying $1,000,000 for the tickets, but they don't address detecting or preventing the problems listed above.

There is also a large literature on extensibility and programmability. In end-user programming environments (Smith et al 1994; Eisenberg & Fischer 1994), a task domain is formalized so that the desired programs can be expressed as simple combinations of domain-relevant building blocks. This leads to clear programs that are relatively easy to validate and, hence, trust. However, the success of the approach requires completeness, correctness, and composability of the building blocks, so each such environment requires a significant development effort and must focus on a narrow domain. This makes it at best only a partial solution to the problem of extensibility in a general assistant-like interface.

In programming by demonstration (Cypher 1991), the assistant constructs a program for the task by observing and generalizing from the user's actions in performing the task on particular example inputs. Thus, one would expect that one could trust the assistant to behave correctly when given the same (or substantially similar) inputs. However, any inputs falling either outside the union of the given example cases or in their pairwise intersections are unvalidated generalizations and hence potential sources of error. In the ticket-buying example, we might demonstrate (a) buying acceptably priced grand tier tickets, and (b) buying acceptably priced balcony tickets. But such demonstrations do not address what to do when both types are acceptably priced, nor what to do when neither is. Any action taken by an automatically generated behavior would have a good chance of being undesirable to the user. A related approach is that of "learning interface agents" (Mitchell et al 1994; Maes 1994), where an assistant again generalizes task heuristics by watching the actions of a user. This suffers from the same potential problem of unvalidated generalization.

Another approach to acquiring assistant behaviors is through *knowledge sharing* (Hall 1995a; Maes 1994) between different assistants or agents. In these approaches, task behaviors are assembled by acquiring knowledge of how to complete the various subtasks from other assistants. For example, a ticket-buying behavior might be assembled from a top-level plan obtained from one agent, together with previously acquired knowledge of how to manipulate a phone line, as well as knowledge of what phone number to dial for the ticket agency obtained from a directory. In these approaches, the trust problem is compounded not only by the worry of sharing with malicious or incompetent users, but also by the fact that what is

desirable to one user may be undesirable to another. For example, while I may be happy accepting second balcony tickets, you may not.

In the AI-planning approach (Etzioni & Weld 1994), a task is specified declaratively (typically in first order logic) and a planner constructs a plan using axiomatized domain knowledge via automated reasoning. While the planner may guarantee the correctness of the plan implementation with respect to the stated task specification, the problem here (leaving aside the computational complexity of the planning problem) is the difficulty in stating a complete and correct formal specification that includes enough detail to account for graceful degradation and all the unexpected situations that can arise during execution. It is hard to rule out plans which correctly implement the task specification but violate some overriding (but unstated) user goal. Etzioni and Weld give the example of planning a way to free up disk space (a desirable goal) by deleting the user's essential files (an undesirable plan).

Thus, virtually no matter how the assistant obtains the knowledge of how to carry out a task, one must address the problem of how the user is to gain trust in the correctness of that knowledge.

Approaches to the Trust Problem

Testing-based approaches to trust. Maes (Maes 1994) and Cohen, et al (Cohen et al 1994) adopt an informal testing-based approach to trust: it is assumed that the longer the user observes the assistant working, either on actual tasks (Maes) or in simulation (Cohen, et al), the better able is the user to predict its behavior. However, the user has no *rigorous* basis for predicting the assistant's behavior on unseen cases. This trust "growth" is based on the human user making (usually unwarranted) generalizations from specific instances of the assistant's behavior. Moreover, as knowledge is added to the assistant, either through acquiring new capabilities (e.g., in Maes's memory-based architecture) or through fixing errors in existing behaviors/knowledge, its behavior can change, calling into question trust based on informal past experience.

Trust in end-user programming environments. As stated earlier, powerful end-user programming environments tend to result in relatively small, clear programs that are readily understood and, hence, trusted. The limitation of this approach is the necessity of narrow domain specificity and large development effort to enable building the tool in the first place. However, when a task falls within the coverage of an existing tool, this is an attractive approach.

Trust in generated plans. In the AI-planning approach to behavior acquisition, Weld and Etzioni (Weld & Etzioni 1994) propose making specifications easier to trust by automatically incorporating safety, tidiness, and thriftiness knowledge into the planner's axioms; however, this has yet to be demonstrated in a practical implementation. It seems to require a detailed, personalized logical model of harm. This need for personalization would seem make it difficult to reuse the "harm knowledge" that makes up these axioms between individuals, since different individuals have different conceptions of harm and desirable behavior.

Specification proof. Another approach, adapted from software engineering methodology, is to have the user state the properties desired of the behavior as a formal specification, and then use automatic or semi-automatic theorem proving to show that the acquired behavior satisfies the specification (see, e.g., (Gilham et al 1989)). The advantage of this approach is that one has high confidence (much higher than is (or should be) achieved through informal testing) that the task knowledge will carry out the specified behavior; however, the disadvantages include (a) the notorious difficulty people have in formally and completely stating what they want, (b) the complexity of automated theorem proving, and (c) the fact that one can't use the behavior in practice until a sufficiently complete specification is proved correct. For example, knowing that the behavior will never overspend its budget or that it will never deadlock does not guarantee that it will buy desirable tickets with that budget (or indeed even that it will buy tickets at all).

Note that all of these approaches share the problem that (a) executing untrusted or partially trusted behaviors autonomously can lead to harm to the user, and (b) they all require significant effort to achieve full trust, if such is even possible (which it isn't for testing-based approaches). Thus, in practice very few behaviors will ever be fully trusted, and yet executing incompletely trusted behaviors can lead to harm.

The Incremental Validation Approach to Trust

In a paper currently under review (Hall 1995c), I propose a novel approach to trust, called incremental validation, that attempts to solve this problem. It is applicable to behaviors acquired by any of the aforementioned techniques, as long as the behavior itself can be expressed (possibly via translation) as a reactive system (Hall 1995b).

Description of the approach. The *incremental validation approach* has the following components.

- A formal definition of a *behavior* for a task as the knowledge or code that the assistant uses to carry out the task, viewed as a reactive system (Hall 1995b).
- A formal representation for a proof that a behavior acts in accord with the user's desires (and hence is trustable). This representation is termed a *trust library* and supports the incremental acquisition of a proof and the use of this proof to support

- *Guarded execution* of untrusted or partially trusted behaviors: essentially, a partially trusted behavior can be safely executed autonomously by the assistant until a situation is encountered that lies outside the set of trusted situations. At that time, the assistant executes a predefined *contingency behavior*, which may do anything, but is typically used either to (a) abort the behavior in as cheap and safe a way as possible, or (b) ask the user to explicitly okay the next action determined by the behavior, in which case the behavior is resumed. (Cypher's EAGER system (Cypher 1991) has a guarded execution mode, but every step must be checked by the user, since there is no explicit trust representation. Hence, there is no trust growth, reuse, or coverage reporting either.)
- An approach to *incremental trust growth* based on sound scenario generalization (Hall 1995b). Essentially, each time the assistant encounters an untrusted situation during guarded execution, it is remembered and soundly generalized. The user is subsequently asked whether whenever a formally similar situation arises a similar action should be taken. If so, this general situation/action pair is entered into the trust library, thereby broadening the set of trusted behaviors. Subsequently, guarded execution succeeds on a larger fraction of input cases.
- A technique for *trust reuse* that allows populating a new trust library for a behavior that has been evolved (or bug-fixed) from an old behavior by transferring trust knowledge from the trust library of the old behavior.
- Tools for computing and displaying *trust coverage and noncoverage*; these reports provide guidance to the user who wishes to construct new situations that might not have yet arisen in practice, so they may be generalized and entered into the trust library.

The incremental validation approach can be seen as falling between the informal testing approach and the specification proof approach in that it is incremental and based on concrete cases (which are relatively easy for a user to comprehend and whose desirability is relatively easy to determine) and yet it results in proved, quantified statements about the behavior.

As described, the incremental validation approach is useful only for behaviors that are used several times, such as message filtering behaviors, electronic purchasing behaviors, etc. If the behavior is only intended to be used once (e.g., an interactive email message), it must be "handheld" at every step (i.e., the user must okay each action before it is carried out). One solution is to combine incremental validation with an initial simulation phase (Cohen et al 1994), using formal trust growth *during simulation* instead of at execution time. The behavior can be launched once the noncoverage report shows adequate trust coverage.

A central question for any trust approach is how fast trust grows (as a function of the amount of experience (e.g., number of test cases) the user has with the behavior). In my paper (Hall 1995c), I prove that for any operational profile (Musa 1993) (distribution of inputs encountered in practice) trust grows expenontially fast: the probability of encountering an untrusted situation in practice is less than k^n, where $0 \leq k < 1$ and n is the number of tests. I also show that this is a weak upper bound in practice.

Limitations and Future Work. Interactive handholding may introduce unacceptable response delays into behaviors subject to time constraints. For example, a behavior that must answer an incoming phone call cannot wait too long for the user to okay its action, lest the caller hang up. In the early stages of trust growth, when most steps have to be checked, the user must be available and attentive for such behaviors.

An open question surrounds the practicality of asking users to okay generalized transition descriptions. There is likely some class of relatively expert users who will be comfortable and accurate in this, as the approach seems to capitalize on strategies similar to those used by expert programmers in comprehending code (Koenemann & Robertson 1991). Moreover, human bosses, parents, and teachers are capable of delegating to and debugging their human assistants and pupils, but how well this transfers to an automated assistant remains open. This suggests the need for both research into presentation techniques and usability studies.

The approach is limited to behavior representations supporting incremental formal validation. (Such support may include automatic translation steps, such as that from BL.2 (Hall 1995a) to EBF (Hall 1995b).) The current prototype can accomodate finite-state, rule-based (EBF), and simple procedural (BL.2) behavior representations, but it is not obvious how to extract reactive-system style descriptions of single behaviors when the assistant is implemented, e.g., as a monolithic neural net or case-based reasoning system.

Conclusion

While delegating mundane, idiosyncratic tasks to an assistant-like interface is highly desirable, we must develop ways for users to build trust in assistant behaviors. With the possible exception of end-user programming environments, which are highly domain-specific and therefore brittle, all methods of acquiring task knowledge are susceptible to the problem of trust when tasks to be automated put significant assets at risk. For most approaches to trust, it is either impractical or impossible to achieve full trust in a behavior, and yet there is no means provided for safely executing partially trusted behaviors. The incremental validation approach addresses these problems by providing guarded execution, trust growth and reuse, and coverage reporting facilities. Note that in increasing the trustworthiness of the assistant, the approach appears

to work against autonomy, since the user must supervise the operation of untrusted behaviors. However, the exponential rate of trust growth means that the assistant quickly becomes autonomous over most of its operational profile.

Thus, of the three critical problems that must be solved in order to realize assistant-like interfaces (autonomy, trust, and behavior acquisition), trust has received the least attention in the research literature. This paper has argued that the problem is critical, because highly personalized and idiosyncratic task knowledge must be acquired directly from the user whose needs it addresses, so such behaviors will not be as well supported and documented as commercial software packages. While critical, I believe that the incremental validation approach shows that the trust problem is surmountable.

References

(Cohen et al 1994). P. Cohen, A. Cheyer, M. Wang, & S.C. Baeg; An open agent architecture; In *Proc. 1994 AAAI Spring Symposium on Software Agents*, 1–8, 1994, AAAI. On WWW: `ftp://ftp.ai.sri.com/pub/papers/cheyer-aaai94.ps.gz`.

(Cypher 1991). A. Cypher; EAGER: programming repetitive tasks by example; in *Proc. 1991 ACM Conf. on Human Factors in Computing Systems*, 431–437; 1991.

(Eisenberg & Fischer 1994). M. Eisenberg & G. Fischer; Programmable design environments: integrating end-user programming with domain-oriented assistance; in *Proc. 1994 ACM Conf. on Human Factors in Computing Systems*, 431–437; 1994, ACM Press.

(Etzioni & Weld 1994). O. Etzioni & D. Weld; A softbot-based interface to the internet; *Comm. ACM, 37(7)*, 72–76, 1994.

(Gilham et al 1989). L. Gilham, A. Goldberg, and T.C. Wang; Toward reliable reactive systems; Kestrel Institute Tech. Memo. KES.U.89.1 (1989).

(Hall 1995a). R.J. Hall; Agents helping agents: issues in sharing how-to knowledge; in *Proc. 1995 AAAI Spring Symposium on Information Gathering from from Heterogeneous, Distributed Environments*, 65–68.
`http://www.isi.edu/sims/knoblock/sss95/hall.ps`.

(Hall 1995b). R.J. Hall; Systematic incremental validation of reactive systems via sound scenario generalization; *J. Automated Software Engineering 2(2)*, 131–166, 1995; Norwell, MA: Kluwer Academic.

(Hall 1995c). R.J. Hall; Trusting your assistant; (currently under review; available from author).

(Koenemann & Robertson 1991). J. Koenemann & S.P. Robertson; Expert problem solving strategies for program comprehension; in *Proc. 1991 ACM Conf. on Human Factors in Computing Systems*, 125–130; 1991, ACM Press.

(Maes 1994). P. Maes; Agents that reduce work and information overload; *Comm. ACM, 37(7)*, 30–40, 1994.

(Mitchell et al 1994). T. Mitchell, R. Caruana, D. Freitag, J. McDermott, & D. Zabowski; Experience with a learning personal assistant; *Comm. ACM, 37(7)*, 80–91, 1994.

(Musa 1993). J. Musa, Operational profiles in software-reliability engineering; *IEEE Software 10(2)*, 14–32, 1993.

(Norman 1994). D. Norman; How might people interact with agents; *Comm. ACM, 37(7)*, 68–71, 1994.

(Riecken et al 1995). D. Riecken, P. Maes, B. Shneiderman, & D.C. Smith; Dealing with complexity: uniting agents and direct manipulation (panel); in *Conf. Companion of 1995 ACM Conf. on Human Factors in Computing Systems*, 191–192, 1995, ACM Press.

(Shneiderman 1991). B. Shneiderman; Demonstrational interfaces: coming soon? (panel discussion, remarks of B. Shneiderman); in *Proc. 1991 ACM Conf. on Human Factors in Computing Systems*, 396; 1991, ACM Press.

(Smith et al 1994). D. C. Smith, A. Cypher, & J. Spohrer; KIDSIM: programming agents without a programming language; *Comm. ACM, 37(7)*, 55–67, 1994.

(Sun 1995). Sun Microsystems; Java Home Page; Available on WWW: `http://java.sun.com/` (1995).

(Wayner 1994). P. Wayner; Agents away (article on General Magic's Telescript), *Byte*, May 1994, 113–116.

(Wayner 1995). P. Wayner; *Agents Unleashed*, Chapter 12: Safe-TCL; 1995; Chestnut Hill, MA: AP Professional.

(Weld & Etzioni 1994). D. Weld & O. Etzioni; The first law of softbotics; In *Proc. 12th Natl. Conf. on AI*, 1994, AAAI Press.