IaDEA: A Development Environment Architecture for Building Generic Intelligent User Interface Agents

Scott M. Brown
†Department of Electrical and Computer Engineering
Air Force Institute of Technology
Wright-Patterson AFB, OH 45433–7765 USA
{sbrown,sbanks,mstytz}@afit.af.mil

Eugene Santos Jr.
Computer Science and Engineering
University of Connecticut
Storrs, CT 06269–3155 USA
eugene@eng2.uconn.edu

Sheila B. Banks† and **Martin R. Stytz†**

Abstract

The need exists in the work force for generic intelligent user interface agents to address the problem of increasing taskload that is overwhelming the human user. Interface agents could help alleviate user taskload by providing abstractions and intelligent assistance in a self-contained software agent that communicates with the user through the existing user interface and also adapts to user needs and behaviors. The benefits of a generic intelligent user interface agent environment is it can be applied to any highly interactive and information intensive software system from freight and parcel management systems to Wall Street financial investment and analysis.

We desire to address the two following difficulties with developing interface agents: (1) The extensive number of existing computer systems makes it impractical to build these agents by hand for each system; (2) Any such agent must be compliant with existing user interface standards and business practices (e.g., the United States Air Force's Common Operating Environment (COE) standards as defined by the Defense Information Infrastructure (DII)). Thus, an environment for constructing intelligent interface agents that facilitates standards compliance is necessary.

We investigate the feasibility of developing an integrated environment for constructing generic intelligent user interface agents. We base our proposed design on three components: (1) a knowledge acquisition tool for support of specification and design of interface agents taking in to account compliance with existing computer-based systems and user interface standards; (2) a utility theory-based methodology for interface agent requirements, assistance, and adaptivity; and (3) a correction model for user adaptivity based on user metrics evaluation.

Keywords: intelligent user interfaces, agent development environments, user adaptive interfaces

Introduction

As computer-based technology continues to progress and the increase of information-based paradigms in modern work force operations continues, solutions for these operations must be capable of delivering increased amounts of information for the human user. Unfortunately, as the amount of information increases, the user's task to process this information has become overwhelming. Furthermore, for human operators to just simply use the interface to these highly complex computer systems currently requires highly specialized training and education that is both lengthy and costly to businesses.

The need exists for systems that can help solve this problem by providing abstractions and intelligent assistance in a self-contained software agent that communicates with the user through the user interface. Since each user operates in a different fashion, processing information in their own way, a complete intelligent interface agent must be capable of adapting to user needs and behaviors.

We desire to address the two following difficulties with developing interface agents: (1) The extensive number of existing computer systems makes it impractical to build these agents from scratch for each system; (2) The agent must be compliant with existing user interface standards and business practices. Thus, an environment for constructing intelligent interface agents that satisfies standards compliance is necessary.

Prior Related Work in Agent Development Environments

We describe several existing related projects for developing adaptive intelligent user interfaces. The need for development tools to aid designers in the specification and design of agent-based systems is critical. However, agent development environments are still relatively immature. Although each is similar to the approach we present here with respect to at least one area of our approach, none present the overall approach we use.

Several commercial and research projects attempted to address the problem of agent development (Cohen *et al.* 1994; Martin, Cheyer, & Lee 1996; Barbuceanu & Fox 1996; IBM 1997; Microsoft 1997). Essentially, all the projects provide a visual development environment, with associated methods specifying agent "behaviors" via an agent communication language. Most

provide a simple syntactic check on the agent's specification. The commercial tools focus mainly on the communication aspects of agent development and, like human-computer interaction researchers, the presentation of the agent to the user. This presentation has, to date, been in the form of anthropomorphized agents, complete with "faces" and "voices." We discuss the two most promising research environments for interface agent development. As will be discussed later, the agent development environment problem is still an active research area.

Open Agent Architecture

The goal of the Open Agent Architecture project (Cohen *et al.* 1994; Martin, Cheyer, & Lee 1996) is to develop an open agent architecture and accompanying user interface for networked desktop and handheld machines. Their system supports multiple agents for distribution of users' tasks and interoperability between application subsystems. To that end, their agent building environment tool allows users to generate code "stubs" for several popular programming languages. Their system includes two other tools — the Linguistic Expertise Acquisition Program (LEAP), for interfacing a new agent with existing linguistic support agents such as natural language parsers and speech recognition systems, and PROJECT, for creating and maintaining repositories of reusable agents.

The Open Agent Architecture's tools allow designers to specifying the agents' interfaces in an extension of Prolog. Agents define their capabilities (called *solvables*) and inform a facilitator agent of those capabilities. The facilitator is responsible for delegating tasks based on the capabilities. The use of a Prolog-like agent communication language allows the facilitator to reason over the capabilities of the agents, i.e., determine if its invocation conditions are true and what the needed parameters for invocation are. This reasoning is only about the agent's explicitly defined interface, and does not consider its internal capabilities. This limitation precludes the possibility of user-modeling and user-intent prediction in any agent developed by this environment.

Agent Building Shell

Barbuceanu, *et al.* (Barbuceanu & Fox 1996) address issues related to actual programming constructs for internal representation and reasoning of agents and the development of tools to support how an agent represents its "view" of the world, to include updating this representation based on interaction within the environment. The Agent Building Shell (ABS) (Barbuceanu & Fox 1996) contribution to the field of agent development environments is the attention given to an agent's internal knowledge representation and reasoning about the environment. The authors use rule-based descriptions, called conversation plans, of agent actions in a given situation. A conversation plan specifies the available conversation rules, their control mechanism and the local data-base that maintains the current state of the conversation. Conversations can be represented graphically with finite state automata. Error recovery rules can be specified to deal with incompatibilities among the current state and incoming messages and are invoked when a conversation rule cannot handle the current situation. Rules are specified with a coordination language. The authors assume a deterministic environment. For their environment, this assumption is reasonable. However, for interface agent development, where the domain (including the user's behavior within the domain) is anything but deterministic, we must chose knowledge representations capable of representing uncertainty. ABS rules must also be specified by the user.

Interface Agent Development Environment Architecture

The main problem with existing agent building environments is they fail to be more than non-agent code development tools with a means of specifying a communication protocol. Most environments focus on the communication aspect and distribution of a task to multiple agents. Although one strength of agents is the ability to distribute tasks to specialized agents for handling those tasks (e.g., e-mail agents, calendar agents, information retrieval agents), focusing just on the communication aspects ignores other aspects of agent specification and development such as adaptivity and robustness. All systems described here suffer from the same weaknesses — lack of environment specification, adaptivity mechanisms, and agent knowledge representation and reasoning mechanisms. Additionally, these systems do not adequately deal with human-agent interaction. The underlying difference between agent-based and software engineering is the dynamics of the environment in which the agents must "perform," to include the dynamic needs of the user. We must address these issues explicitly within our development methods, tools, principles, etc.

To address the concern that agent development environments are little more than software engineering tools, we identify the following three areas where agent development environments have been deficient.

Environment Specification. Existing environments implicitly address the environment where an agent must perform. An explicit representation of the environment can bring to the fore-front critical domain considerations and enable the designer to identify those domain environment features (e.g., stimuli, affectors and human-factor considerations) with which the interface agent must deal. Since we are concerned solely with interface agents, we also desire to specify user behavior within the environment. The use of an explicit specification of the domain environment for agent-based development is a technique used in software engineering (Rumbaugh *et al.* 1991).

Adaptivity Mechanisms. None of the current en-

vironments explicitly address the fact that agents typically operate in dynamic environments and, therefore, must be capable of adaptive behavior. While adaptivity requirements can be addressed at the code level, they are better addressed at the specification level, where we can determine what to adapt, how to adapt it, when to adapt it, and why.

Agent Knowledge Base and Reasoning Mechanisms. All but IBM's Agent Building Environment and the Agent Building Shell fail to provide a persistent knowledge base and underlying reasoning mechanism for agents to reason about their environment via sensing, acting, and reacting. The other systems merely provide a way for the agent to react to an explicit activation; Autonomous agent behavior, where the agent can reason about the actions of the user in an attempt to autonomously act on the user's behalf, is not addressed. IBM's solution is to provide sensing "adapters" and simple if-then rules to control the overall actions of the agent. Agents perform their task(s) if the antecedent of their "activation" rule is satisfied. The knowledge base is stored in a library for use by all agent(s).

Current agent development environments typically focus on collaboratitive, autonomous multi-agent system specification and development (more the former than the latter). However, they tend to ignore the fact that many agents, in particular interface agents, need the ability to adapt to the changing needs of the user and environment. Furthermore, as we mentioned earlier, we wish to develop generic intelligent user interface agents usable in many different domains. Generic interface agent development requires a robustness not found in other development environments. We address these issues explicitly within our proposed agent development architecture, the Intelligent Agent Development Environment Architecture (IaDEA) as depicted in the figure below.

A number of the components of our architecture have already been completed. In particular, the Core Interface Agent Architecture (CIaA) and associated adaptivity requirements metrics are complete and we have reported on them elsewhere (Brown *et al.* 1998; Brown, Santos Jr., & Banks 1998). The target system application programmers' interface (API) and the interface standards are determined *a priori* by the existing system. The main component of IaDEA yet to be implemented is the ASLAN environment. We discuss our proposal for ASLAN next.

Agent Specification Language Environment

To address the short-comings of current agent development environments, we propose the Agent Specification LANguage (ASLAN) environment. To one degree, our proposal supports our existing Core Interface Agent Architecture (CIaA) and enables developers to easily perform the needed initial knowledge acquisition for CIaA. To another degree, our proposal extends, where possible, existing methods and tools to expand their capabilities and coverage of agent specification and design issues. In some areas, we propose adding new functionality, in others, we merely propose extending existing functionality to account for adaptive interface agents. ASLAN will be capable of incorporating user interface standards (e.g., the Defense Information Infrastructure standards for a Common Operating Environment) as an independent module that automatically drives agent interface developments. This methodology allows for automatic and "hands-free" updating of ASLAN as user interface standards themselves are updated and change.

We propose to use the Knowledge Query and Manipulation Language (KQML) (Mayfield, Labrou, & Finin 1996) and Ontolingua (Gruber 1993) with a knowledge acquisition tool front-end. KQML is both a message format and message-handling protocol, supporting runtime sharing of knowledge between heterogenous (and homogeneous) agents. KQML is emerging as a standard within the agent community and there exists a number of tools (to include application programming interfaces, or APIs) to support KQML integration into an agent development environment and implementation.

Ontolingua is a language for representing ontologies. An ontology is a specification of a representational vocabulary for a shared domain of discourse — definitions of classes, relations, functions, and other objects. In relationship to agent research, ontologies describe the concepts and relationships that can exist for an agent or a community of agents. Gruber (Gruber 1993) uses ontologies for the purpose of enabling knowledge sharing and reuse between agents. An ontological commitment is an agreement (by the agent) to use a vocabulary (i.e., ask queries and make assertions) in a way that is consistent with respect to the theory specified by an ontology. While Ontolingua is good for representing and sharing knowledge, it is poor for acquiring that knowledge due to its notation and the inherent complexity of designing ontologies. We therefore propose to "shield" the designer from Ontolingua by providing the ASLAN environment tool. Knowledge acquired via the tool can then be encoded into a KQML application programming interface using KQML's performatives and Ontolingua's content format for sharability with the agent research community at large and use within an interface agent-based system.

Addressing Agent Development Environment Deficiencies Figure 1 shows that an agent specification language may be derived for a given software system from user interface standards, the target systems application program interface (API), and specifications required by the human interface developer. We now describe how the ASLAN environment can be used to address the above three deficiencies of existing environments.

IaDEA Environment Specification. To specify our environment, we must specify what events we expect from the user interface, i.e., the stimuli the user interface can provide to the agent. For example, the interface agent needs to be able to determine when a

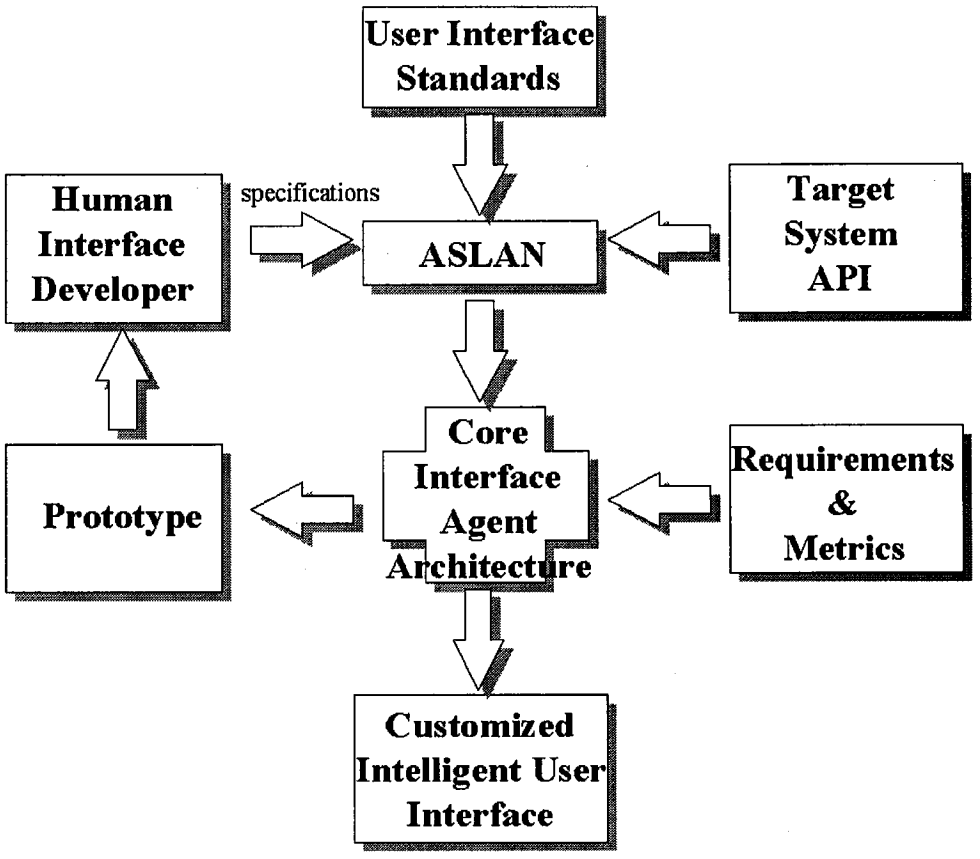

FIGURE 1. Proposed Interface Agent Development Environment Architecture (IaDEA): High-level process flow for construction of customized intelligent user interfaces.

button is pressed, a menu item is selected, the user exits the system, etc. Additionally, the agent must know how it can affect the environment it is in, i.e., the affectors. These affectors can either be defined by the target system's API or we can assume that the designer of the agent will have direct access to the target system's source code. ASLAN can be used to elicit an explicit enumeration of the "call-backs" the agent can use as input from and output to the application. This enumeration is sufficient to capture the application specific stimuli and affectors.

Interface agent development must explicitly deal with one component other agent systems possibly ignore — the human agent. Human users have beliefs, desires, and intentions, abilities, preferences, emotions, plans, and goals. To offer beneficial assistance to the user, we must be able to capture these characteristics. Furthermore, the environment the interface agent and human user are situated in has a direct impact on the user's behaviors as well as the environment being affected by these two agents. This interaction between environment, user, and interface agent must be captured.

User modeling is concerned with how to represent users' knowledge and interaction within a system to adapt the system to the needs of users. Proliferation of user modeling as a means of accurately capturing the beliefs, abilities, and intent of users is apparent. Researchers from the fields of artificial intelligence, human-computer interaction, psychology, education, and others have all investigated ways to construct, maintain, and exploit user models.

Vassileva feels we need to start using lessons learned from user modeling to impact the way we view interactive human-computer environments (Vassileva 1997). She discusses viewing these environments along three orthogonal dimensions: elements — the goals, plans, resources, and actions composing the atomic entities an agent (human or otherwise) is concerned with, processes — the types of processing (e.g., reaction, deciding, learning) that takes place in an agent, and relationships — the way agents interact with one another. Her approach makes explicit reasoning about the purpose of adaptations, treats human and computer agents the same in the environment, takes into account user motivation, emotions, and moods, and presents a unified model of collaborative, cooperative, and adverse behavior.

The ability to accurately capture user characteristics and completely address Vassileva's viewpoints on interactive human-computer environments is difficult. For

example, there is typically not a specific user action within the environment that determines a certain user characteristic (e.g., intentions). However, several techniques from the user model research field are applicable, e.g., user profiles and stereotypes.

User profiles can be used to represent background, interests, and general knowledge about a user that is typically static. User profiles may be elicited from users by, for example, standardized tests, surveys, and/or assessments. These elicitation techniques are used to construct the user profile. For example, Csinger *et al.* use a simple form-filling operation to elicit interest metrics for "intent-based" authoring (Csinger & Poole 1996; Csinger, Booth, & Poole 1994). Van Veldhuizen *et al.*, use a "skills vector" to define an agent's ability to perform certain tasks, measured by standardized scores (Van Veldhuizen, Lamont, & Santos Jr. 1998). Additionally, some researchers are investigating ways to incorporate human factors into user models (Mulgund & Zacharias 1996; Schäfer & Weyrath 1997; Stefanuk 1997; Gavrilova & Voinov 1997).

Furthermore, we can elicit various stereotypical user models (Rich 1983). Stereotypes represent "typical" users, representing various traits, characteristics, and attributes that often co-occur in people. Users classify themselves as belonging to a particular stereotype. User modeling researchers, as well as other research disciplines, have used stereotypes and profiles for various purposes (Kay 1994). User modeling examples include using stereotypes to filter World Wide Web documents (HTML/text), basing a user's inferred interests on user adapted stereotypes acquired via experts (Ambrosini, Cirillo, & Micarelli 1997), deriving initial proficiency estimates on a user's level of advancement for use in computer-assisted language learning (Murphy & McTear 1997), and adapting Web-based hypermedia based on stereotypical users' abilities (e.g., disabled and the elderly) (Fink, Kobsa, & Nill 1997). We feel the work presented here and in previous separate work has begun addressing many of Vassileva's concerns (Brown, Santos Jr., & Banks 1998).

In particular, we are interested in capturing the user's intent. Representing the user's intent within the environment is paramount if the interface agent is to predict this intent and therefore offer assistance to the user. Benyon and Murray state "failure to recognize the intentions underlying some user action will result in less satisfactory interaction" as a result of failing to recognize the pursuit of one goal versus another (Benyon & Murray 1993). Brown, Santos Jr., and Banks (1998) state that an approach to predicting user intent is to identify the salient characteristics of the domain environment and specifically determine goals a user is trying to achieve. This approach is based on the belief that what a user intends to do in an environment is the result of environmental events and the goals they are trying to achieve via reaction to stimuli. Therefore, there is a causal relationship between environmental stimuli, human factors, users' goals, and the actions users perform.

As a first step towards modeling user characteristics applicable to the user, the ASLAN environment can function as a knowledge acquisition front-end to Ontoligua to specify both a user profile for a specific user and user stereotypes for classes of users. Any of the aforementioned techniques for specifying user profiles and stereotypes is suitable for use within IaDEA. With regards to specifying user intent, since we contend there exists a causal relationship between environmental stimuli, human factors, users' goals, and the actions users perform, ASLAN can use a directed acyclic graph to show this causality. Furthermore, since ascribing user intent is inherently uncertain, we can assign probabilities to the arcs in the directed graph. We have used this approach in previous research (Brown, Santos Jr., & Banks 1998).

IaDEA Adaptivity Mechanisms. As previously mentioned, a key distinguishing factor between software and agent-based engineering is that agent-based engineering deals with dynamic environments. However, as is evident from the existing agent development environments, adaptivity to dynamic environments is not often explicitly considered. We desire to bring the requirement of adaptivity to the forefront of agent specification and design, guiding the designer to areas in the design where adaptivity is warranted. We discuss our incorporation of adaptivity mechanisms into IaDEA in the next section.

IaDEA Agent Knowledge Base and Reasoning Mechanisms. Since the ASLAN environment uses Ontolingua, and since Ontolingua is not meant as a representation language for reasoning, we should, for efficiency's sake, choose a different knowledge representation for reasoning. The ontology defines an adaptive intelligent agent, its environment (a user interface), and the user's goals and associated actions to achieve those goals within that environment. Since our knowledge base is probabilistic, we need a representation that can handle probabilistic reasoning. Additionally, as we mentioned previously, a causal relationship exists between environmental stimuli, users' goals, and actions. This causality can be used to construct Bayesian networks (Pearl 1988) based on the observable events and goals within the environment (Jameson 1995). Bayesian networks provide a mathematically correct and semantically sound model for representing uncertainty that provides a means to show probabilistic relationships between random variables — in this case goals, actions, pre- and post-conditions.

Core Interface Agent Architecture

With input/output user interface standards and requirements — such as DII-COE compliance and those imposed by the particular software system we are constructing the interface for — fully specified in the ASLAN environment, we can separate out surface issues from the core issue of how such an interface agent should perform. In particular, we propose in this section what an intelligent interface agent architecture

should be and how it should function.

We proceed to give a short presentation of the Core Interface Agent Architecture (CIaA) to describe its interdependency with ASLAN. Each component of CIaA is defined in detail elsewhere (Brown *et al.* 1998; Brown, Santos Jr., & Banks 1998). Figure 2 gives us an overview of CIaA including user intent prediction and continual adaptivity. We delegate the task of ascribing user intent to the interface agent component of the architecture, while continual adaptation of the interface agent's user model is a task shared by the interface and and a collection of correction adaptation agents.

Figure 3 shows the architecture of the interface agent and the correction adaptation agents. The content of the KQML messages is specified via ASLAN. Each target system observation (environmental stimuli, user action (as interpretted via the user interface of the target system), and human factor) can be communicated to the agents via the KQML message passing API. Every observation is stored by the agents' evaluator in a history stack (i.e., most recent observation is on the top of the stack). These observations can be used by the agents as evidence into the user model, represented as a Bayesian network. The interface agent offers assistance via suggestions to the target system (user) by calculating the expected utility of offering assistance for a goal, $EU(\alpha_G)$. $EU(\alpha_G)$ is calculated by performing Bayesian network belief updating on the goal random variable and the utility of suggesting the actions used to achieve the goal, $U(G, a, \Delta)$.

There are three underlying methodologies utilized in CIaA. These methodologies are a utility theory-based approach, requirements metrics, and our correction model and associated correction adaptation agents. In the remainder of this section, we define the utility theory-based approach, requirements metrics, and the correction model.

Development of a Utility Theory-Based Approach for Interface Agent Assistance As we have previously mentioned, ASLAN takes into account the salient characteristics of the domain environment and goals a user is trying to achieve. As Brown, Santos Jr., and Banks (1998) state, there are several advantages to representing users' intentions via goals, such as the following:

- Goal abstraction allows us to design and detect higher level goals, in pursuit of lower level goals.
- Evidence can be easily and intuitively added and removed (in the form of pre- and post-conditions) as a user interacts with the system.
- Pre- and post-conditions for goals and actions are explicitly stated.
- Keyhole plan recognition[1] is made easier by explicitly enumerating atomic actions composing goals (Albrecht *et al.* 1997; Waern 1996).
- Natural language explanations of actions based on prediction of goals can be easily generated.

As Figure 3 shows, CIaA uses a Bayesian network as the underlying knowledge representation for the user model. Bayesian networks' sound mathematical basis in probability theory makes them ideal to represent the uncertainty found in trying to predict a user's intent. It should be noted that knowing the probability a user is pursuing a goal is important, but for an interface agent to be truly useful, we must concern ourselves with the utility of offering assistance to the user. This approach has been used to good affect by several researchers (Horvitz & Barry 1995; Karagiannidis, Koumpis, & Stephanidis 1996; Horvitz 1997; Brown, Santos Jr., & Banks 1998). We can determine the expected utility of offering assistance for a goal, based not only on the probability of an action as determined by performing belief updating of the Bayesian network, but on the utility of performing the action given the user is pursuing the goal.

The general idea is that the interface agent suggests the goal with the greatest expected utility. Utility theory, using Bayesian techniques for assessing the probabilities, is a non-ad hoc approach for predicting user intent. The utility function can take into account *relevancy* of the goal with respect to any number of metrics and/or discriminators in the environment, including human factors identified by ASLAN. These metrics tell us what is important, explicitly enumerating those factors that impact the utility of choosing the goal. Details related to the construction of the Bayesian network and associated utility functions can be found in Brown *et al.* (Brown, Santos Jr., & Banks 1998).

Requirements Metrics and Utiltiy Function We believe it is absolutely essential to develop concrete, measurable requirements for the interface agent. These requirements measure the interface agent's effectiveness in meeting the needs of the user. Without these requirements, any adaptations we make to the user model (via our adaptivity mechanisms) are nothing more than a ad hoc, quixotic attempt to improve the interface agent's ability to provide beneficial assistance. Put another way: if we don't know where we are going, how will we know when we get there?

We believe it is a necessity to first develop concrete, measurable requirements and then use these metrics to determine the effectiveness of an interface agent within an environment. We believe the following interface agent requirements are necessary (but possibly not sufficient):

- **adaptivity** — the ability to modify an internal representation of the environment through sensing of the environment to change future sensing, acting, and reacting for the purpose of determining user intent and improving assistance
- **autonomy** — the ability to sense, act, and react over

[1] Plan recognition is the task of ascribing intentions about plans to an agent (human or software), based on observation of the agent's actions. With heyhole plan recognition, the agent is unaware of or indifferent to the plan recognition process.

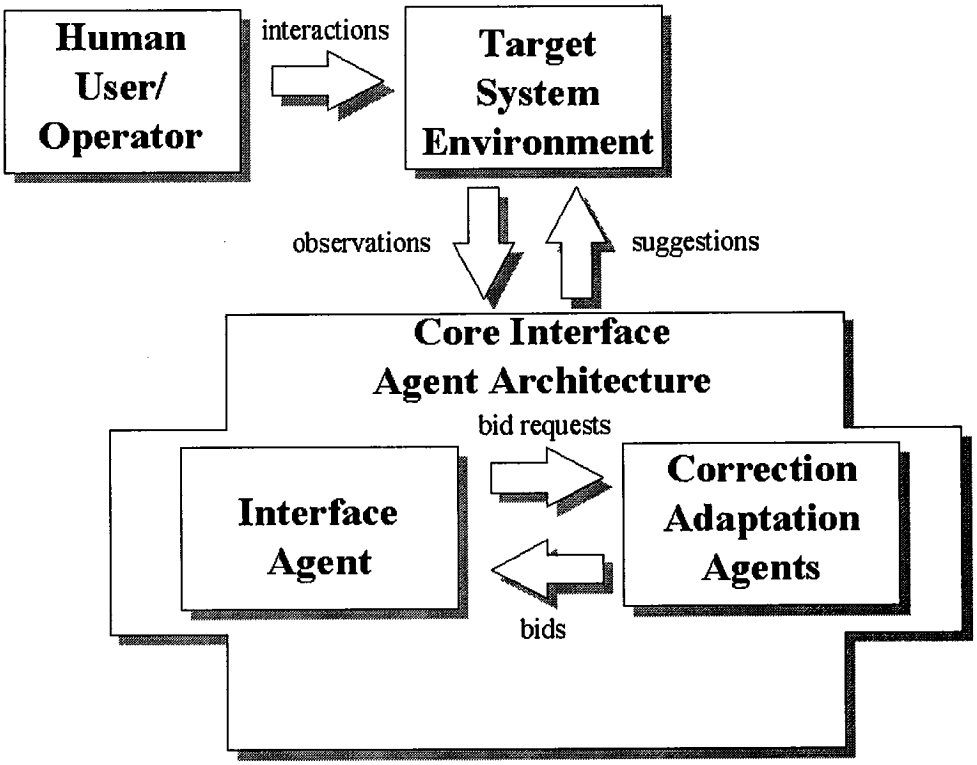

FIGURE 2. Core Interface Agent Architecture (CIaA): Process flow diagram for user intent prediction and continual adaptation of an intelligent user interface agent.

time within an environment without direct intervention

- **collaboration** — the ability to communicate with other agents, including the user, to pursue the goal of offering assistance to the user
- **robustness** — the ability to degrade assistance gracefully.

We have developed an associated set of requirement metrics to measure the effectiveness of the interface agent in meeting these requirements. Details on the requirement metrics set may be found elsewhere (Brown *et al.* 1998).

Using the aforementioned requirements metrics, we can define a requirements utility function that determines the interface agent's utility of meeting the requirements. The utility function $U_{requirements}$ is defined for the requirement metrics of the agent, weighted with respect to their importance, based on some previous history. That is,

$$U_{requirements} : \omega^n \times R^n \times H \mapsto \Re, \qquad (1)$$

where for each history $h \in H$ of previous actions and events, $\omega \in [0,1]$ is a weighting factor for each of the n requirement metrics R, and the utility function maps to a real number. The higher the value of the utility of our interface agent, the more "successful" it is in meeting its requirements. This utility function allows us to identify which requirements are not being met and attempt to correct the problem by altering the user model. We can readily use ASLAN to present these metrics to the designer and allow the designer to chose which metrics are appropriate and also determine the relative importance of the metrics with respects to one another, thereby determining the value of each ω_i.

Correction Models and Adaptation Agents Using the ASLAN environment, an interface designer specifies the salient characteristics of the environment. This specification, along with the target system API and any applicable user interface standards are used to generate the Core Interface Agent Architecture. However, due to the inherent dynamics of complex systems and the uncertainty in accurately representing user intent, our agent must be capable of adapting over time. This adaptation allows the interface agent to maintain an accurate user model.

Using the aforementioned requirements metrics and requirements utility function, the utility of the user model can be measured to determine how closely the user model is to the real (exhibited) behavior of the user. Since the user's behavior may change over time, deviating from the original user model specified via the ASLAN environment (assuming we accurately captured the user's model initially), any adaptation techniques must be used *over time* to make sure the user model reflects the user's behavior.

This proposed approach is implemented as a multi-agent system for adaptation of the interface agent's user

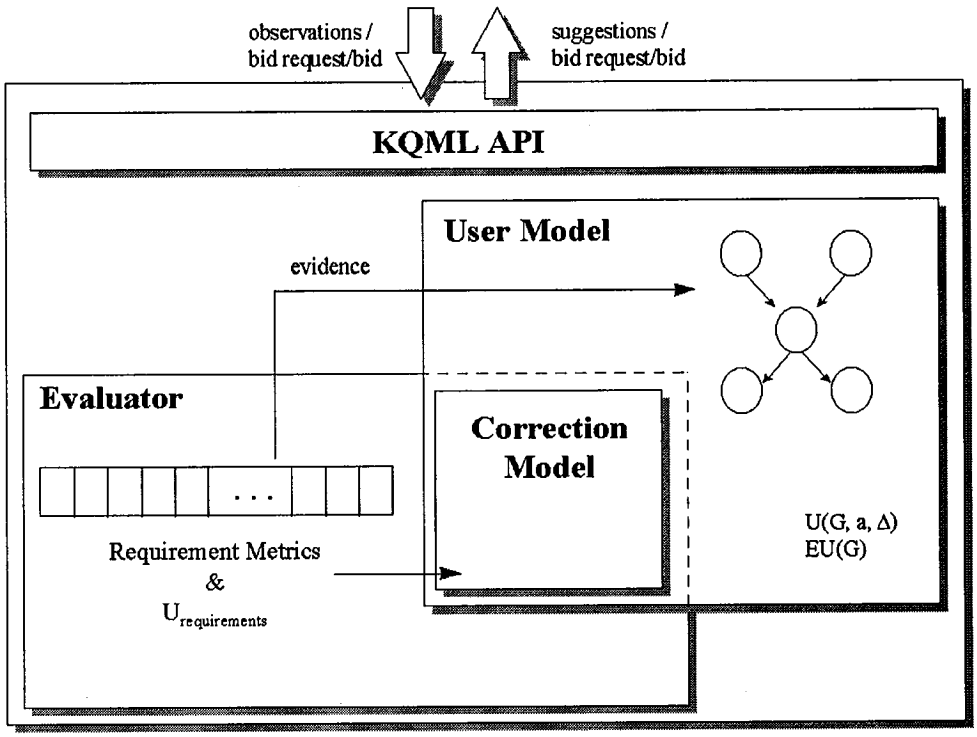

FIGURE 3. Interface Agent and Correction Adaptation Agent Architecture.

model (Brown *et al.* 1998). See Figure 2 and Figure 3. The approach to correcting the user model is to have the interface agent request "help" via a *bid request* from *correction adaptation agents* — specialized agents capable of correcting problems with the user model by adapting it to improve the interface agent's requirements utility. The adaptation mechanism is encapsulated in the correction adaptation agent's correction model. The correction adaptation agents suggest changes to the ailing interface agent user model in accordance with a *correction model* based the concept of a contractual bidding process. These correction adaptation agents engage in a "bidding process" to recommend changes to the ailing interface agent. The interface agent serves as a *manager* agent, responsible for determining when a contract is available, announcing the contract to be filled, and receiving bids from the *bidder* agents. The interface agent evaluates the *utility* of the bids based on the correction adaptation agent's ability to improve the requirements utility function value. The correction adaptation agent that can improve the interface agent's requirements utility function value the most "wins" the contract to correct the user model.

Conclusion

The need for development methodologies for agents is readily apparent. Our approach to interface agent development is to use the Interface agent Development Environment Architecture to explicitly specify our environment, to include the target system API, applicable user interface standards, and human factor concerns, model the user using a utility theory-based approach and probability theory, and adapt the user model over time to accurately reflect the user's interaction with the environment. We do not mean to presuppose that our approach is the Holy Grail of interface agent development. As we begin to implement the approach presented here, we are sure to uncover unforeseen problems and pitfalls. However, we believe our approach and the ASLAN environment will address deficiencies of current agent development environments — environment specification, adaptivity, and agent knowledge base and reasoning mechanisms. We desire to easily and directly incorporate user interface standards into ASLAN to produce compliant intelligent interface agents. In other words, we can treat user interface standards as simply an input database to define ASLAN. As a first step towards realizing the ASLAN environment tool, ASLAN will be developed to support the existing core interface agent architecture, requirements, metrics, and correction adaptation agents.

We have proposed how we can define the "surface-level" details of agent development in ASLAN and then use the concepts of utility theory, requirements and metrics, as well as a multi-agent system approach to develop a core interface agent architecture. The use of utility theory, combined with a knowledge representation capable of capturing the inherent uncertainty and dynamics of user modeling, is a non-ad hoc approach to user intent prediction. Furthermore, the use of ex-

plicit probabilistic causal goals and actions allows us to represent our approach in any number of graphical knowledge representations, such as Bayesian networks or influence diagrams and, as stated previously, probabilistic approaches have additional advantages. The keyhole plan recognition approach is valid and has been used in other related systems (Albrecht *et al.* 1997; Waern 1996). We attempt to improve on their work by providing knowledge acquisition tools (i.e., ASLAN) for designers to specify what actions to take, when to take them, why to take an action, and how to take it. This improved approach also has merit and is being investigated by other projects (Karagiannidis, Koumpis, & Stephanidis 1996). We improve on their work by extending their approach to agent-based environments, with heterogeneous information sources (e.g., correction adaptation agents), and provide a mathematically sound approach to uncertainty. Our initial results indicate this approach works (Banks *et al.* 1997).

References

Albrecht, D. W.; Zukerman, I.; Nicholson, A. E.; and Bud, A. 1997. Towards a bayesian model for keyhole plan recognition in large domains. In Jameson, A.; Paris, C.; and Tasso, C., eds., *Proceedings of the Sixth International Conference on User Modeling (UM '97)*, 365–376. SpringerWien New York.

Ambrosini, L.; Cirillo, V.; and Micarelli, A. 1997. A hybrid architecture for user-adapted information filtering on the World Wide Web. In Jameson, A.; Paris, C.; and Tasso, C., eds., *User Modeling: Proceedings of the Sixth International Conference, UM97.* Vienna, New York: Springer Wien New York. 59–61. Available from http://um.org.

Banks, S. B.; Harrington, R. A.; Santos Jr., E.; and Brown, S. M. 1997. Usability testing of an intelligent interface agent. In *Proceedings of the Sixth International Interfaces Conference (Interfaces 97)*, 121–123.

Barbuceanu, M., and Fox, M. S. 1996. The architecture of an agent building shell. In Woolridge, M. J.; Müller, J. P.; and Tambe, M., eds., *Intelligent Agents II: Agent Theories, Architectures, and Languages*, 235–250. Berlin: Springer.

Benyon, D., and Murray, D. 1993. Adaptive systems: from intelligent tutoring to autonomous agents. *Knowledge-Based Systems* 6(4):197–219.

Brown, S. M.; Santos Jr., E.; Banks, S. B.; and Oxley, M. E. 1998. Using explicit requirements and metrics for interface agent user model correction. In *Proceedings of the Second International Conference on Autonomous Agents (Agents '98).* to appear.

Brown, S. M.; Santos Jr., E.; and Banks, S. B. 1998. Utility theory-based user models for intelligent interface agents. In *Proceedings of the Twelfth Canadian Conference on Artificial Intelligence (AI '98).* to appear.

Cohen, P. R.; Cheyer, A. J.; Wang, M.; and Baeg, S. C. 1994. An open agent architecture. In *AAAI Spring Symposium*, 1–8.

Csinger, A., and Poole, D. 1996. User models and perceptual salience: Formal abduction for model recognition and presentation design. In *Proceedings of the Fifth International Conference on User Modeling (UM '96)*, 51–58.

Csinger, A.; Booth, K. S.; and Poole, D. 1994. AI meets authoring: User models for intelligent multimedia. *Artificial Intelligence Review* 8:447–468.

Fink, J.; Kobsa, A.; and Nill, A. 1997. Adaptable and adaptive information access for all users, including the disabled and the elderly. In Jameson, A.; Paris, C.; and Tasso, C., eds., *User Modeling: Proceedings of the Sixth International Conference, UM97.* Vienna, New York: Springer Wien New York. 171–173. Available from http://um.org.

Gavrilova, T., and Voinov, A. 1997. An approach to mapping of user model to corresponding interface parameters. In *Proceedings of the Embedding User Models in Intelligent Applications Workshop*, 24–29. held in conjunction with the Sixth International Conference on User Modeling (UM '97).

Gruber, T. R. 1993. A translation approach to portable ontology specifications. *Knowledge Acquisition* 5(2):199–220.

Horvitz, E., and Barry, M. 1995. Display of information for time-critical decision making. In *Proceedings of the Eleventh Uncertainty in Artificial Intelligence*, 296–305.

Horvitz, E. 1997. Agents with beliefs: Reflections on Bayesian methods for user modeling. In Jameson, A.; Paris, C.; and Tasso, C., eds., *User Modeling: Proceedings of the Sixth International Conference, UM97.* Vienna, New York: Springer Wien New York. 441–442. Available from http://um.org.

IBM. 1997. IBM agent building environment (ABE): A toolkit for building intelligent agent applications. World Wide Web Page http://www.networking.ibm.com/iag/iagsoft.htm.

Jameson, A. 1995. Numeric uncertainty management in user and student modeling: An overview of systems and issues. *User Modeling and User-Adapted Interactions* 5:193–251.

Karagiannidis, C.; Koumpis, A.; and Stephanidis, C. 1996. Deciding 'what', 'when', 'why', and 'how' to adapt in intelligent multimedia presentation systems. In Faconti, G., and Rist, T., eds., *Proceedings of the Twelvth European Conference on Artificial Intelligence Workshop "Towards a Standard Reference Model for Intelligent Multimedia Presentation Systems".* John Wiley & Sons, Ltd.

Kay, J. 1994. Lies, damn lies, and stereotypes: pragmatic approximations of users. In *User Modeling: Proceedings of the Fourth International Conference, UM94.* 175–184. Early draft of invited keynote presented at UM'94 available at http://www.cs.su.oz.au/ judy/Research/index.html.

Martin, D. L.; Cheyer, A.; and Lee, G. L. 1996. Agent development tools for the open agent architecture. In *Proceedings of the First International Conference on the Practical Application of Intelligent Agents and Multi-Agent Technology*, 387–404. London: The Practical Application Company Ltd.

Mayfield, J.; Labrou, Y.; and Finin, T. 1996. Evaluation of kqml as an agent communication language. In Woolridge, M. J.; Müller, J. P.; and Tambe, M., eds., *Intelligent Agents II: Agent Theories, Architectures, and Languages*, 347–360. Berlin: Springer.

Microsoft. 1997. Microsoft agent. Available at `http://www.microsoft.com/intdev/agent/`.

Mulgund, S. S., and Zacharias, G. L. 1996. A situation-driven adaptive pilot/vehicle interface. In *Proceedings of the Third Annual Symposium on Human Intercation with Complex Systems*, 193–198.

Murphy, M., and McTear, M. 1997. Learner modelling for intelligent CALL. In Jameson, A.; Paris, C.; and Tasso, C., eds., *User Modeling: Proceedings of the Sixth International Conference, UM97.* Vienna, New York: Springer Wien New York. 301–312. Available from http://um.org.

Pearl, J. 1988. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference.* San Mateo, CA: Morgan Kaufmann.

Rich, E. 1983. Users are individuals: Individualizing user models. *International Journal of Man-Machine Studies* 18:199–214.

Rumbaugh, J.; Blaha, M.; Premerlani, W.; Eddy, F.; and Lorenson, W. 1991. *Object-Orient Modeling and Design.* Prentice Hall.

Schäfer, R., and Weyrath, T. 1997. Assessing temporally variable user properties with dynamic Bayesian networks. In Jameson, A.; Paris, C.; and Tasso, C., eds., *User Modeling: Proceedings of the Sixth International Conference, UM97.* Vienna, New York: Springer Wien New York. 377–388. Available from http://um.org.

Stefanuk, V. L. 1997. Embedding user models in intelligent interfaces. In *Proceedings of the Embedding User Models in Intelligent Applications Workshop*, 18–23. held in conjunction with the Sixth International Conference on User Modeling (UM '97).

Van Veldhuizen, D. A.; Lamont, G. B.; and Santos Jr., E. 1998. Comparing computer generated military actors with specific skills. In *Proceedings of the Society of Photo-Optical Instrumentation Engineers (SPIE) Aerospace/Defense Sensing and Controls Workshop on Modeling and Simulating Sensory Response for Real and Virtual Environments.* to appear.

Vassileva, J. 1997. A new view of interactive human-computer environments. In Jameson, A.; Paris, C.; and Tasso, C., eds., *User Modeling: Proceedings of the Sixth International Conference, UM97.* Vienna, New York: Springer Wien New York. 433–435. Available from http://um.org.

Waern, A. 1996. *Recognising Human Plans: Issues for Plan Recognition in Human-Computer Interaction.* Ph.D. Dissertation, Royal Institute of Technology.