Designing a Universal Interfacing Agent

Darsana P. Josyula*, Michael L Anderson†, Don Perlis*,†

*Department of Computer Science
†Institute for Advanced Computer Studies
University of Maryland, College Park, MD 20742
{darsana, anderson, perlis}@cs.umd.edu

Abstract

This paper argues the need for a universal interfacing agent that users can use to control different task-oriented systems. It discusses the different capabilities required of such an agent and what is required for implementing those capabilities. In addition, it provides an approach for its implementation

1. Introduction

Device interfaces are Janus-faced, with one face looking outward and interacting with the user and the other looking inward and controlling the device. The actual tasks that a device performs determine the inner face of the interface. One of the most common strategies applied for designing the outer face of the interface is to focus on these tasks, relying on familiar controls for each task. For instance, simple video players have controls for play, rewind, forward and stop corresponding to the four functions that the player can perform. The advantage of this strategy is that the capabilities of the device are easily accessible to the user and hence, in effect the user controls the device directly. However, the number of controls on the outer face of the interface increases in proportion to the number of capabilities. Therefore, as devices get more complex, it becomes increasingly difficult to display all the capabilities without making them resemble airplane cockpits.

Another common strategy employed to design the outer face is to focus on the user needs, providing controls for what the designer perceives to be the most common user needs. In this strategy, there is a clear mapping layer that provides the proper mapping from the outer face controls to the inner face device functions; wherein a control on the outer face may map to more than one capability of the device. Thus, in some video players there is a control that automatically rewinds the tape before it plays the tape. The advantage of this strategy is that the users have easy access to some commonly required behavior. However, the behavior that the designer perceives to be the most common, might not match some users' requirements since each user is unique and the behavior that users require from a device differ. For the same reasons, it is impossible to foresee all possible user-specfic needs. Even if a designer succeeds in foreseeing all possible user-specific needs and providing access controls that can trigger behaviors that satisfy those needs, the resulting outer face will become too complex for any practical use.

A third strategy employed is to allow flexible outer faces that users can adapt to suit their needs. For instance, some CD players allow users to program the tracks to be played repeatedly. Here, the mapping layer between the outer face and the inner face is not fixed by the designer, rather it is programmed by the user. Therefore, the user chooses the set of inner face device functions that are to be mapped to each outer face control, and the interface allows and keeps track of such user programmed mappings.

In fact, when flexible outer faces are allowed, the interface as a whole, can be thought of as an agent that (i) keeps track of the current mappings in the mapping layer, (ii) translates the outer face user selections to appropriate sets of device capabilities and (iii) allows changes in the mapping layer through the outer face. The amount of dynamic mapping change that is facilitated depends on how sophisticated the particular interfacing agent is; the more sophisticated the agent is, the more intelligent the resulting device is considered to be. Examples of such intelligent devices include wheel chairs (Gomi and Griffith, 1998; Levine et al., 1999), construction vehicles (Gomi and Ide, 1999) and Portable Satellite Assistant(PSA) (Gawdiak et al., 2000).

The advantage of having flexible outerfaces is that the users have additional flexibility in operating the device in a manner that suits their needs. As Kay (Kay, 1990) points out, interfacing agents[1] can revolutionize computing since

This research was supported in part by grants from AFOSR and ONR.

[1]The intelligent interfacing agents that are in use today are mainly for web related applications. For instance, intelligent information agents like search engines (e.g., Google, HotBot), news watchers (Billsus and Pazzani, 1999) and browsing assistants (Lieberman, 1997) find, evaluate and filter information based on the user's personal interests. Another class of intelligent interfaces concentrate on cooperation with users or other agents to find solutions to complex problems. These agents have been applied to provide help in the areas of organizing

a user need not manipulate a system directly, but can indirectly control a system by interacting with the agent. The disadvantage is that one has to learn to program a device in the manner one desires, before one can operate it in that fashion.

Today, each device interface is designed as part of a device itself. The disadvantage of this is two-fold. First, there is very little in common between the outer faces of the different device interfaces. Hence, as users acquire more electronic devices (cameras, cell phones, PDAs, etc.) they may find learning each new interface (outer face) increasingly burdensome. And second, the inner face of an interface is very closely knit with the underlying device functionalities and hence they are more-or-less inseparable. Therefore, the interfacing agent designed for one device cannot be easily reused to interface another device.

As consumer electronics become more complex and various, designers will need to consider the possibility of employing a single, shared, general and flexible interfacing agent, which has an outer face that users can easily learn to interact with, a mapping layer that can be modified easily and an inner face that can be adapted to control many different devices. An interfacing agent equipped with a natural language outer face and having appropriate mechanisms to alter the mapping layer and the inner face will serve this goal. Users can tailor the behavior of different task-oriented systems (TOS) to suit their needs using such a universal interfacing agent.

For example, consider a pool controller that accepts commands to heat the pool, stop heating and provide temperature of the water. To maintain the temperature of the pool at $90^o F$ between 8:00 pm and 9:00 pm if it is a working day, and between 10:00 am and 1:00 pm if it is a non-working day, a user directly interacting with the pool controller has to (i) keep track of the time (ii) keep track of the temperature by observing the temperature reading periodically during the interval in which the temperature has to be maintained and (iii) issue commands to heat and stop heating based on the observed temperature. On the other hand, a user interacting with an interfacing agent can instruct the agent to maintain a temperature of $90^o F$ between 8:00 pm and 9:00 pm if it is a working day and between 10:00 am and 1:00 pm if it is a non-working day; the agent can the issue commands at appropriate times to heat and stop heating based on the temperature, time and whether the user is working or not on a particular day. Thus, by integrating a rational interfacing agent with a task-oriented system, the user gains the flexibility to adapt the system to meet his/her unique requirements.

email (Lashkari et al., 1994), shopping online (Chavez and Maes, 1996), scheduling meetings (Haynes et al., 1997), automating tasks (Hoyle and Lueg, 1997) and providing advice (T.Selker, 1994).

2. Capabilities

Devices perform tasks in response to commands (e.g., pressing a button) [2] and produce results either in response to commands or automatically. A device interface essentially provides a mechanism for the user to access the different tasks or results.

A universal interfacing agent (UIA) is feasible as a device interface if the interfacing logic is separable from individual device functionalities. If such a separation is possible, then interfacing with a new device may involve merely providing the device dependent information to the UIA; i.e., specifying the device specific tasks, readings, commands and constraints to the UIA. An intuitive list of such device-independent interfacing capabilities is given below[3]:

2.1. Communicating With Device

A UIA needs to have the ability to send valid commands to the particular device to which it is connected, in order to initiate appropriate tasks. Some tasks may have parameters associated with them. For instance, the task of heating the pool may have an associated temperature parameter that specifies the temperature to which the pool needs to be heated. A UIA needs to be able to issue commands that have such parameters in order that it can interface with different kinds of devices. Besides, it has to be able to receive information or accept (observe) the readings that a given device produces.

2.2. Communicating With User

A UIA has to keep track of user specified goals, preferences and constraints. Moreover, it has to request clarifications and provide relevant information to the user.

2.3. Scheduling Device Actions

The different actions that a device can do include performing a device task or producing a device reading. A key interfacing capability is to allow users to schedule these actions. Therefore, any UIA needs to be able to handle scheduling requests. Examples of possible scheduling requests include the following:

- perform actions immediately. e.g., Heat pool now. Read temperature now.
- perform actions after a specific amount of time. e.g., Heat pool after 5 minutes. Read temperature after 5 minutes.
- perform actions at specific time. e.g., Heat pool at 7:00 am. Read temperature at 7:00 am.

[2] We are not interested in those tasks that are performed automatically since the user (or the UIA) cannot initiate such tasks.

[3] Initial results from our user survey on the most desired features for different task-oriented systems seem to substantiate this list.

- perform actions at periodic intervals, indefinitely. e.g., Heat pool at 6:00 am everyday. Heat pool at 10:00 am on non-working days.

- perform actions at periodic intervals, for a specific number of times. e.g., Heat pool at 8:00 am everyday this week. Heat pool at 10:00 am on every non-working day this month.

- perform actions when certain conditions hold. e.g., Heat pool at 7:00 pm today if the short-circuit problem has been fixed by then. Heat pool after 6:00 pm if the temperature is below 80^oF and the voltage level is not low.

2.4. Composing Complex Commands

Users differ in the set of tasks that they desire to be associated with each outer face control. They may even differ in the manner in which the tasks are to be grouped together. For instance, some tasks may have to done in sequence while others may have to done concurrently. For tasks that are to be performed in sequence, the time delay between tasks may vary. In order to provide the users the ability to perform user-desired tasks in a more effective manner, the UIA has to permit users to create new composite commands.

Complex commands can be composed in different ways. Commands, that cause primitive device tasks to be performed, may be combined together in various ways to create composite complex commands. These composite commands may be combined with either primitive commands or other complex commands to create other complex commands. Each complex command may also include the scheduling information for the different commands that constitute it; thus providing users the capability to compose new commands that cater to the behaviors that they need from a device, and thus, tailor each device to meet their needs. The various scheduling possibilities are similar to the ones discussed in Section 2.3.

2.5. Canceling Commands

Another important device-independent interfacing capability is to provide users the capability to cancel previously issued commands. In the simplest case, the command to be cancelled may not have been acted upon (as in, an action to be done in future); however, in more complicated cases, the action may have been just started, in the process of completion or even completed.

2.6. Tolerating Perturbations

Since the user is insulated from directly manipulating, controlling and maybe even monitoring the target TOS, much of the proactive responsiveness which would normally be the provenance of the user must be taken over by the agent. Thus, it becomes the responsibility of the agent to respond to any perturbations such as contradictory information or a difference between expected and actual outcomes, quickly and effectively. To detect such perturbations, the UIA has to keep track of the effects of the TOS commands it issues, by interpreting the results that the TOS produces, and when such results are not available, by confirming with the user if and when required. That is, if the TOS has the necessary sensors to observe the results of the tasks it perform, then the agent can keep track of the effect of the TOS commands it issues, by interpreting these TOS results; but if the TOS does not have such sensors, then the agent can rely on confirmations that it gets from the user.

3. Requirements

What is required of a UIA, to achieve the capabilities listed in the previous section, can be summed up as interpret user requests and act based on those requests. In order to act according to the user requests for performing device tasks or producing device results or readings, a UIA has to represent and reason about, the tasks that a device can perform and the effects of those tasks. Issues associated with representing and reasoning about actions include – specifying persistence of properties (Shoham, 1988; McCarthy and Hayes, 1969), qualifications for performing actions (McCarthy, 1980), ramifications of actions, time and duration of actions, time and duration of effects, overlapping, concurrent, natural and indeterministic actions (Reiter, 1996) and relationships between actions.

The UIA also needs to have a concept of time in order to permit scheduling requests. Particularly, a concept of the current time (now), how it evolves as time passes and how it relates to the standard measures of time (minutes, seconds etc.) are necessary.

In addition to representing and reasoning about actions and effects as well as time and its passage, the UIA needs to represent and reason about the different domain objects (device parts) that are involved in the tasks or results that the device performs or produces. Issues associated with representing and reasoning about domain objects include—specifying properties of objects, type and range of values for properties, object relationships and categories as well as object names and equivalences.

All user requests are not directed towards performing a primitive device task or producing a primitive device result; rather they may be directed towards composing complex commands from existing commands and issuing tasks associated with complex commands. For the UIA to be able to properly interpret user requests for composing complex commands, it has to have the ability to learn or accept more information into its knowledge base during its operation. In addition, it needs to be able to access and use this new information in its ongoing operation, so that users gain the flexibility to trigger the behaviors they want from the TOS, by issuing these complex commands.

To permit users to schedule different activities, the UIA has to make commitments for performing these activities at the scheduled time, and plan its actions accordingly. For instance, if the user request is to maintain some property of an object at some value for a specific period of time, the UIA needs to observe the value during the specified period and if the value of the property changes, then it needs to trigger the device tasks that will bring the property back to the required value.

Moreover, in order to detect perturbations, a UIA has to track the effects of the tasks that it initiates, either by observing these effects by issuing appropriate commands to the given device or by confirming with the user, when necessary. Also, since perturbations can appear as contradictions between an old belief and a new belief, it has to tolerate contradictions as well.

4. Approach

One way of creating such an interfacing agent is to implement the the mental attitudes of beliefs, desires, intentions, expectations and achievements within a time-sensitive and contradiction-tolerant logical framework that allows rich knowledge representation as well as flexible knowledge modification. Active logic (Elgot-Drapkin and Perlis, 1990; Purang, 2001) is a formalism that allows rich, flexible, time-sensitive and contradiction-tolerant knowledge base management and reasoning.

Over the past few years, we have been working on creating a universal interfacing agent within the Active Logic framework. The details of the implementation are discussed in (Josyula et al., 2004; Josyula et al., 2005). The basic idea there is that the UIA interprets the user requests and creates desires to accomplish the requests. Based on its knowledge and availability of time and resources, the agent creates an intention to perform the desired action. Since the agent does not interact with the real world, the only way it can determine whether the command that it issued has resulted in the desired action is by confirming it from either the user or the TOS. In order to make this determination, the agent creates an expectation, regarding the outcome, based on available knowledge, whenever it initiates an action. The agent keeps track of those desires, intentions and expectations that are achieved, achievable and unachievable in order to take appropriate action. For instance, in case a desire cannot be achieved the user may need to be informed.

The current implementation of our agent ALFRED has been successfully interfaced with the following domains:

- Simulated Pool: ALFRED controls the temperature settings of a simulated pool based on user needs.
- Movie Player: ALFRED plays different movies based on user requests.
- Toy Train: ALFRED moves various trains to different cities based on user requests in a toy train domain.
- Simulated House: ALFRED controls different appliances in a simulated house based on user needs.
- Home Designer: ALFRED helps the user create and move different objects in a house model.
- Chess Player: ALFRED plays chess for the user by sending the user requested moves to a chess program.

To switch from one domain to another, ALFRED has to be loaded with domain specific information like names of the domain objects and the syntax and semantics of the valid TOS commands in that domain.

5. Conclusion

As Kay (Kay, 1990) points out, interfacing agents can revolutionize computing since a user need not manipulate a system directly, but can indirectly control a system by interacting with the agent. The intelligent interfacing agents that are used today are mainly for web related applications. For instance, intelligent information agents like search engines (e.g., Google, HotBot), news watchers (Billsus and Pazzani, 1999) and browsing assistants (Lieberman, 1997) find, evaluate and filter information based on the user's personal interests. Another class of intelligent interfaces concentrate on cooperation with users or other agents to find solutions to complex problems. These agents have been applied to provide help in the areas of organizing email (Lashkari et al., 1994), shopping online (Chavez and Maes, 1996), scheduling meetings (Haynes et al., 1997), automating tasks (Hoyle and Lueg, 1997) and providing advice (T.Selker, 1994).

This paper argues the need for a universal interfacing agent that users can use to control different task-oriented systems. It discusses the requirements for such an agent and provides an approach for its implementation.

6. References

Billsus, Daniel and Michael J. Pazzani, 1999. A Personal News Agent that Talks, Learns and Explains. In Oren Etzioni, Jörg P. Müller, and Jeffrey M. Bradshaw (eds.), *Proceedings of the Third International Conference on Autonomous Agents (Agents'99)*. Seattle, WA, USA: ACM Press.

Chavez, Anthony and Pattie Maes, 1996. Kasbah: An agent marketplace for buying and selling goods. In *Proceedings of the First International Conference on the Practical Application of Intelligent Agents and Multi-Agent Technology (PAAM-96)*. Practical Application Company.

Elgot-Drapkin, Jennifer and Don Perlis, 1990. Reasoning Situated in Time I: Basic Concepts. *Journal of Experimental and Theoretical Artificial Intelligence*, 2(1):75–98.

Gawdiak, Yuri, Jeff Bradshaw, Brian Williams, and Hans Thomas, 2000. R2d2 in a softball: the portable satellite assistant. In *Proceedings of the Fifth International Conference on Intelligent User Interfaces*.

Gomi, Takashi and Ann Griffith, 1998. Developing intelligent wheelchairs for the handicapped. In V. Mittal, H. Yanco, J. Aronis, and R. Simpson (eds.), *Assistive Technology and Artificial Intelligence*, volume 1458 of *LNAI*. Springer-Verlag, pages 150 – 178.

Gomi, Takashi and Koichi Ide, 1999. The development of an autonomous field transport vehicle with an active vision system. In *Proceedings of the Society of Instrument and Control Engineering Conference (SICE'99)*.

Haynes, T., S. Sen, N. Arora, and R. Nadella, 1997. An automated meeting scheduling system that utilizes user preferences. In *Proceedings of the First International Conference on Autonomous Agents (Agents-97)*. ACM Press.

Hoyle, Michelle A. and Christopher Lueg, 1997. Open sesame: A look at personal assistants. In *Proceedings of the Second International Conference on Practical Applications of Intelligent Agents and Multi-Agent Technology (PAAM'97)*.

Josyula, Darsana P., Michael L. Anderson, and Don Perlis, 2004. Domain-Independent Reason-Enhanced Controller for Task-ORiented systems - DIRECTOR. In *Proceedings of the National Conference on Artificial Intelligence (AAAI-04)*.

Josyula, Darsana P., Michael L. Anderson, and Don Perlis, 2005. Metacognition for dropping and reconsidering intentions. In *Papers from the 2005 AAAI Spring Symposium on Metacognition in Computation*.

Kay, Alan, 1990. User Interface: A Personal View. In Brenda Laurel (ed.), *The Art of Human-Computer Interface Design*. Addison-Wesley Publishing Company, Inc., pages 191–207.

Lashkari, Yezdi, Max Metral, and Pattie Maes, 1994. Collaborative interface agents. In *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94)*. AAAI Press.

Levine, Simon P., David A. Bell, Lincoln A. Jaros, Richard C. Simpson, Yoram Koren, and Johann Borenstein, 1999. The NavChair Assistive Wheelchair Navigation System. *IEEE Transactions on Rehabilitation Engineering*, 7(4).

Lieberman, Henry, 1997. Autonomous interface agents. In *Proceedings of the ACM Conference on Computers and Human Interface (CHI-97)*.

McCarthy, John, 1980. Circumscription - a form of non-monotonic reasoning. *Artificial Intelligence*, 13:27–39.

McCarthy, John and Patrick Hayes, 1969. Some philosophical problems from the standpoint of artificial intelligence. *Machine Intelligence*, 4:463–502.

Purang, K., 2001. Alma/Carne: Implementation of a Time-situated Meta-reasoner. In *Proceedings of the Thirteenth International Conference on Tools with Artificial Intelligence (ICTAI-01)*.

Reiter, Ray, 1996. Natural Actions, Concurrency and Continuous Time in the Situation Calculus. In *Proceedings of the Fifth International Conference on Principles of Knowledge Representation and Reasoning (KR -96)*. Cambridge MA.

Shoham, Y., 1988. *Reasoning About Change: Time and Causation from the Standpoint of Artificial Intelligence*. Cambridge, MA: The MIT Press.

T.Selker, 1994. Coach: a teaching agent that learns. *Communications of the ACM*, 37(7):93–99.