Natürlichsprachliche Bedienung technischer Systeme

Bernd Ludwig, Stefan Mandl, Peter Reiß, Günther Görz, Herbert Stoyan

Lehrstuhl für Künstliche Intelligenz
Friedrich-Alexander-Universität Erlangen-Nürnberg
Vorname.Nachname@informatik.uni-erlangen.de

Abstract: Komplexe elektronische Systeme spielen eine immer größere Rolle im Alltag. Beispiele sind moderne Geräte der Unterhaltungselektronik wie Fernseher mit integriertem DVD-Recorder und DVB-T-Empfänger oder Navigationssysteme im Auto. Auch komplexe Software auf Heim-PCs hat einen großen Funktionalitätsumfang. Durch die fortschreitende Leistungsfähigkeit steigen also die Anforderungen an den Benutzer bei der Bedienung dieser Systeme. Daher bleibt ein großer Teil der Funktionalität unbenutzt, weil der Benutzer gar nicht alle Fähigkeiten des Systems kennt oder nicht weiß, wie die Funktionen zu aktivieren sind. Wir stellen einen Ansatz zur intuitiven Bedienung vor, der natürlichsprachliche Kommandos in Systemaktionen umsetzt, die Ausführung überwacht und flexibel auf Fehler in der Ausführung reagieren kann.

1 Bedienschnittstellen für technische Geräte

Geräte mit komplexer Funktionalität werden heute meist *menügeführt* bedient, auch wenn die Interaktionsmodalität – etwa natürliche Sprache – flexibler sein könnte. In Abbildung 1 ist eine beispielhafte menübasierte Interaktion zwischen einem Nutzer und seinem TV-Gerät zu sehen, hier werden letztlich die Funktionen der Knöpfe auf der Fernbedienung durch Kommandowörter ersetzt. Statt zu wissen, welche Knöpfe er in welcher Reihenfolge drücken muss, um Nachrichten sehen zu können, muss der Benutzer Schlüsselwörter lernen, damit ihn das TV-Gerät „verstehen“ kann.

Menügeführte Bedienung

Welches Gerät wollen Sie bedienen?
– Fernseher
Welche Funktion wollen Sie aktivieren?
– Programmliste
Zu welcher Uhrzeit suchen Sie Sendungen?
– jetzt
Welches Genre?
– Nachrichten
Wählen Sie einen Kanal aus!
– Kanal 2

‚Spontane‘ Bedienung

Ich möchte jetzt Nachrichten sehen.

Abbildung 1: Menübasierte versus spontane Bedienung

Taste	Kontext	Wirkung	Zweck
Fensterheber	Tür zu Zündung aus	hoch oder runter (ruckweise)	andere Fensterstellung
	Tür zu Zündung an, kurzer Druck	Aktivierung der Automatik (hoch oder runter)	
	Tür zu, Zündung an	hoch oder runter (fließend)	

Abbildung 2: Ein elektrischer Fensterheber kann unterschiedliche Wirkungen haben.

Die Wirkung einzelner Schlüsselwörter hängt jedoch oft auch vom aktuellen Gerätezustand ab – genauso wie Bedienknöpfe je nach Gerätezustand unterschiedliche Wirkungen haben. Abbildung 2 stammt aus der Bedienungsanleitung eines aktuellen PKW-Modells und gibt einen Überblick darüber, was das Drücken des Fensterhebers alles bewirken kann. Man kann aber wohl meist davon ausgehen, dass der Fahrer gar nicht um die unterschiedliche Funktionsweise in verschiedenen Kontexten weiß und ob ihm immer zur rechten Zeit einfällt, was er tun muss um sein Ziel zu erreichen. Ein noch deutlicheres Beispiel ist das Kopieren einer nicht standardkonformen Vorlage auf einem handelsüblichen Kopierer. Der Benutzer könnte in einem einzigen Satz ausdrücken, wozu er das Gerät veranlassen will, die Funktion ist aber im Menü versteckt. Ohne genaue Kenntnis der Menüstruktur nutzt auch eine andere Eingabemodalität nichts, solange diese ebenfalls menübasiert arbeitet.

Komfortabler für den Benutzer ist *spontane Bedienung*, wobei er lediglich mitteilt, was er mit dem Gerät tun möchte. Das Beispiel in Abbildung 1 verdeutlicht, dass die spontane Bedienung eines TV-Geräts über Sprache für den Nutzer großen Komfortgewinn bringen kann. Ein derartiges Bedienprinzip technisch so umzusetzen, dass ein Bediensystem an die Anforderungen verschiedenster Geräte ohne großen Aufwand angepasst werden kann, stellt eine erhebliche Herausforderung bei der Entwicklung des Bediensystems selbst, aber auch für die Softwarekomponenten des zu bedienenden Geräts dar.

Im Forschungsprojekt EMBASSI (`http://www.embassi.de`) wurde folgende Systemarchitektur entworfen (siehe Abbildung 3): Die unimodalen I/O-Geräte übersetzen Signale (wie Sprache, Gesten oder Messwerte) in *tokens*, die von der multimodalen Datenaufbereitung interpretiert werden. Die Bedeutung dieser Interpretation im Zusammenhang eines ganz bestimmten Bedienvorgangs zu verstehen, ist die Aufgabe von Dialogmanagement und Assistenzmethoden. Hierin liegt der fundamentale Perspektivenwechsel von der menügeführten zur spontanten Bedienung: im Dialogmanagement wird einer Benutzereingabe ein Ziel zugeordnet, das den Wunsch des Benutzers erfasst; die Assistenzebene sucht nach einer Möglichkeit, das Ziel durch einzelne Schritte zu erfüllen. Jeder einzelne Schritt wird auf der Ausführungsebene erledigt; die Geräte – ein zentraler praktischer Aspekt – melden Fehler an die Assistenz. Dort kann eine andere, durchführbare Lösung gesucht werden.

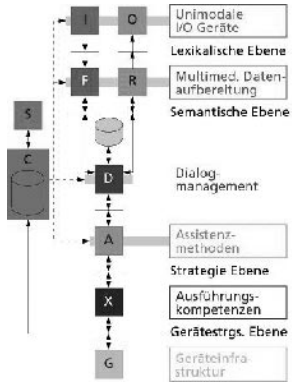

Abbildung 3: Systemarchitektur des EMBASSI-Dialogsystems

2 Konzeptionelle Anforderungen an flexible Bedienschnittstellen

Die Details unseres Lösungsansatzes und ein Überblick über ähnliche Arbeiten anderer Gruppen sind in [Lud06] beschrieben. In diesem Beitrag soll ein Überblick über unsere Arbeiten gegeben werden.

Während Sprachbediensysteme auf dem aktuellen Stand der Technik meist Kommandos aus der Spracheingabe des Benutzers extrahieren, das bisherige Bedienparadigma der Menüauswahl aber beibehalten, soll eine flexible Bedienschnittstelle auch implizite Anweisungen verstehen, auf Rückfragen des Benutzers reagieren und mit dem Anwender angemessene Reaktionen auf unvorhergesehene Ereignisse aushandeln können. Ein Beispiel aus dem Navigationsszenario wäre, dass bei einer Stauwarnung auf der gewählten Route das Navigationssystem eine Umleitung berechnet. Diese wird dem Benutzer dargestellt, ebenso, welche Auswirkungen das für die Fahrtdauer und die Ankunftszeit hat. Schließlich muss der Fahrer bestätigen, ob die Ausweichroute gefahren werden soll oder nicht.

Das Beispiel soll verdeutlichen, dass die Flexibilität einer Bedienschnittstelle nicht alleine aus verschiedenen Eingabemodalitäten resultiert, sondern dass Reaktionsstrategien für unerwartete Ereignisse – seien es nun Fragen des Anwenders oder Fehler im Bedienablauf – den erwünschten Zuwachs an Flexibilität ermöglichen. Grundlegend dafür, in einer Bediensituation eine geeignete Reaktionsstrategie auszuwählen, sind folgende Fähigkeiten der Bedienschnittstelle:

- Fähigkeit, einen regulären Bedienablauf planen zu können (Soll-Definition)
- Fähigkeit, den Kontext jedes einzelnen Bedienschritts über die Werte von Systemparametern charakterisieren zu können (Ist-Definition)
- Fähigkeit, Diskrepanzen zwischen Ist- und Soll-Zustand analysieren und dafür eine geeignete Reaktion festlegen zu können.

```
produce-coffee
:parameters (?c - cup ?j - jura)
:precondition
  (and (under-spout ?c)
       (not (service-request ?j)))
:effect
  (and (not (empty ?c)) (ready ?j))
```

Abbildung 4: Beispiel für einen Planoperator in PDDL

2.1 Reaktionsstrategien

Für die Reaktion auf eine Diskrepanz zwischen Ist- und Soll-Zustand ist jeweils viel Domänenwissen vonnöten. Nichtsdestotrotz gibt es generische Strategien, wie auf neue Situationen reagiert werden kann:

- Verzögern: Oft ist es am besten, noch abzuwarten, ob die Diskrepanz überhaupt relevant wird.
- Delegation: Übertragung der Aufgabe an einen anderen Agenten
- Neuplanung: Der Bedienablauf wie bisher geplant kann nicht realisiert werden. Stattdessen muss ein neuer Plan gefunden werden.
- Relaxation: Oft ermöglicht eine leichte Modifikation des Ziels die Fortführung des Bedienablaufs.
- Verhandlung: Die Modifikation des Ziels soll vom Anwender vorgegeben werden.
- Abbruch: Im schlimmsten Fall muss das Bedienziel aufgegeben werden.

Natürlich muss diese Liste unvollständig bleiben, und jeder einzelne Punkt hat in jeder Anwendung eine unterschiedliche Ausprägung. Wie diese generischen Strategien anwendungsabhängig konkretisiert werden können, bleibt eine offene Frage in der Forschung.

2.2 Planen von Bedienabläufen

Für das Berechnen der Aktionssequenzen, die zum Erreichen des Benutzerziels führen, haben wir uns für einen Ansatz mit automatischem Planer entschieden. Die Vorteile sind klare Beschreibungen der Situation vor und nach ausführen einer Aktion, gegeben durch die Vor- und Nachbedingungen der Planoperatoren. So kann Erfolg oder Misserfolg einer Aktionsdurchführung erkannt werden. Des weiteren ist einfaches Neuplanen bei Misserfolg möglich. Zur Illustration dessen, wie eine Bedienschnittstelle einen interaktiven Bedienablauf planen kann, soll im Folgenden ein Beispiel aus einem sprachgesteuerten System für Transportaufgaben diskutiert werden.

Voraussetzung für die Planung ist eine formalisierte Darstellung der Funktionen, die das System anbietet. Zu diesem Zweck nutzen wir die Planungssprache PDDL. Ein Bespiel für eine Funktionsdarstellung ist in Abbildung 4 gegeben. Der Operator beschreibt eine Funktion des Espressoautomaten, der in unserem Demonstrationssystem integriert ist: Die Funktion `produce-coffee` ist ausführbar, wenn eine Tasse unter dem Ausguss steht und wenn der Automat keine Wartung braucht. Dies sind die Vorbedingungen der Funktion. Nachdem die Funktion ausgeführt und Kaffee hergestellt worden ist, wird sich der Systemzustand verändern: Die Tasse wird nicht mehr leer sein und der Automat ist wieder betriebsbereit. Auf diese Weise werden sämtliche Funktionen, die von der Bedienschnittstelle kontrolliert werden, modelliert.

```
(and (parked cup)
     (mode-osc jura))
```

Abbildung 5: Zielbeschreibung eines Bedienablaufs

```
(and (parked cup)
     (empty cup)
     (ready jura)
     (ready robo))
```

Abbildung 6: Beschreibung des aktuellen Systemzustands

Um einen Bedienablauf zu planen, ist noch eine Aufgabenschreibung notwendig. Sie wird aus Spracheingaben des Benutzers generiert und besteht aus einer logischen Konjunktion von Aussagen über den gewünschten Zielzustand des Systems. Das Bedienziel zur Eingabe *„Bringe mir bitte einen Espresso!“* ist in Abbildung 5 zu sehen.

Die Planung geht vom aktuellen Zustand (siehe Abbildung 6) aus und ermittelt eine Folge auszuführender Gerätefunktionen und Bedienschritte des Benutzers.

2.3 Ausführen von Bedienabläufen

Wenn ein Bedienablauf geplant ist, können seine einzelnen Schritte ausgeführt werden. Vor jedem Schritt prüft die Bedienschnittstelle, ob die Vorbedingungen einer Funktion, wie sie im zugehörigen Planoperator spezifiziert sind, im aktuellen Systemzustand gelten. Trifft dies für alle Vorbedingungen zu, dann wird die Funktion ausgeführt. Das Resultat der Ausführung wird über Sensorwerte interpretiert. Stimmen sie nicht mit den Erwartungen, die der Plan für den Bedienablauf stellt, überein, dann wird ein Fehler signalisiert (siehe Abbildung 7).

2.4 Fehlerdiagnosen

Wie soll sich die Bedienschnittstelle „verhalten“, wenn ein Fehler signalisiert worden ist? Das Hauptziel ist ja, in einer möglichst intuitiv nachvollziehbaren Reaktion das Bedienziel

```
produce-coffee (cup c, jura j) {
  if test(under-spout,c)=false
    signal_error;
  else {
    if test(service-request,j)=true
      signal_error;
    else do produce-coffee, c, j;
  };
  if test(empty,c)=true signal_error;
  else {
    if test(ready,j)=false signal_error;
    else return;
  };
}
```

Abbildung 7: Ausführungsroutine einer Gerätefunktion

```
put-cup-on-spout(cup,jura,robo)
draw-off-osc(cup,jura)
produce-coffee(cup,jura)
go-in-place(train)
take-cup-off-spout(cup,jura,robo)
load-cup-on-waggon(cup,jura,robo,train)
park-cup(cup,jura,robo,train)
```

Abbildung 8: Ein Plan für das Bedienziel in Abbildung 5

immer noch zu erfüllen. Im Kontext des aktuellen Bedienablaufs, der für das hier diskutierte Benutzerziel in Abbildung 8 dargestellt ist, muss eine Entscheidung getroffen werden, die den diagnostizierten Fehler am schnellsten reparieren kann.

Dazu wird die Fehlerdiagnose danach klassifiziert, welchem typischen Fehlerfall sie angehört. Jeder Fehlerklasse sind spezifische Reaktionsstrategien zugeordnet, die anhand der konkreten Diagnose bewerten, welcher Nutzen und welches Risiko entsteht, würden sie in der gegebenen Situation angewendet werden. Die Bedienschnittstelle führt dann diejenige Strategie aus, die mit dem größten Nutzen und dem geringsten Risiko behaftet ist.

3 Grenzen modellbasierter Ansätze

Der zuvor vorgeschlagene Ansatz der modellbasierten Gestaltung der Interaktionsschicht eines technischen Systems leidet — wie alle modellbasierten Ansätze — unter dem Problem der Unvollständigkeit der Modellierung. So sind für jede anwendungsspezifische Modellierung immer Anwendungssituationen denkbar, die nicht geeignet berücksichtigt werden. Im Kaffeeautomatenszenario könnte dies sein: Kaffee- oder Bohnenbehälter sind leer; keine sauberen Tassen mehr da; Filter verstopft; Tasse umgefallen; Kaffee kalt, da der Benutzer ihn längere Zeit stehen lässt; Stromausfall: Systemzustand korrespondiert nicht mit Weltzustand; und vieles mehr.

Ausgeführte Aktionen	**Systemzustand nach** `go-in-place`	**Erwarteter Zustand**	**Fehler-diagnose**
`put-cup-on-spout`	`(ready jura)`		none
`draw-off-osc`	`(ready robo)`		none
`produce-coffee`	`¬(parked cup)`		none
`go-in-place`	`¬(empty cup)`		none
	`¬(robo-loaded robo)`		none
	`(mode-osc jura)`		none
	`(in-place train)`		none
	`(under-spout cup)`		none
Last Action	**Observed state**		
`take-cup-off-spout`	`(under-spout cup)`	`¬(under-spout cup)`	robo could not
	`¬(robo-loaded robo)`	`(robo-loaded robo)`	hold the cup

Abbildung 9: Kontextinformation für die Diagnose eines Fehlers im Bedienablauf

Diese Problematik wird in der Regel von Bedienschnittstellen durch eine kontrollierte graphische Interaktion ausgeblendet. Sprachbediente Systeme können den Wortschatz, den Anwender benutzen, um Fehler wie oben beschrieben zu formulieren, nicht verarbeiten. Nicht nachvollziehbare Reaktionen des Systems, durch die Anwender abgeschreckt werden, sind die Folge. Statt das Problem der Grenzen einer Modellierung zu ignorieren, plädieren wir für eine zusätzliche Komponente in der Bedienschnittstelle, die aktiv das System beobachtet und Hypothesen für das Auftreten nicht modellierter Fehler findet. Der Zweck der Komponente besteht darin, zur Laufzeit des Systems, die Möglichkeiten zur Fehlerdiagnose zu erweitern. Ein Nebeneffekt des Ansatzes liegt darin, dass dem Systementwickler z.B. in Benutzerstudien Hinweise auf Lücken in der Modellierung der Gerätefunktionalität gegeben werden können. Eine solche Lücke nennen wir *unsichtbaren Kontext* einer Modellierung. Darunter verstehen wir all jene informellen Gegebenheiten, welche bei der Formalisierung nicht berücksichtigt wurden.

In [MLSS06] haben wir eine Möglichkeit der Handhabung unsichtbaren Kontexts in einem Online-Lernszenario vorgeschlagen. Das Verfahren erlaubt es, anhand von Beispielen Hypothesen über den Einfluss nicht gemessener Sensorwerte auf den zu lernenden Sachverhalt zu generieren und die unbekannten Größen vorherzusagen.

4 Fazit

Der beschriebene Ansatz zur Modellierung von Bedienschnittstellen ist in mehreren Systemen prototypisch implementiert worden ([Lud04a, Hub04, Lud04b]): einer Transport- und Logistik-Domäne ([HL02]), im Verbundprojekt EMBASSI für eine Heimelektronik-Domäne und im Verbundprojekt FORSIP für eine eCommerce-Anwendung [DL05].

Die bisherigen Arbeiten haben die Hypothese bestätigt, dass der Wissenserwerb als Voraussetzung einer Modellierung eine große Hürde darstellt. Ein computerbasiertes Verfahren zur semiautomatischen Modellierung ist daher die aktuelle Hauptanstrengung unserer

Forschung. Das Lernen unsichtbarer Kontexte ist dazu der erste Baustein.

Literatur

[DL05] Sven Döring und Bernd Ludwig. Personalisierung und Situierung in der Mensch-Maschine-Interaktion. *Künstliche Intelligenz*, Seiten 27–35, 2005.

[HL02] Alexander Huber und Bernd Ludwig. A Natural Language Multi-Agent System for Controlling Model Trains. In *Proceedings AI, Simulation, and Planning in High Autonomy Systems (AIS 2002)*, Seiten 145–149, Lissabon, 2002.

[Hub04] Alexander Huber. *Interaktive agentenbasierte Bedienassistenz – Konzeption und Implementation eines Agents für die Bedienung von technischen Systemen*. Dissertation, Universität Erlangen-Nürnberg, 2004.

[Lud04a] Bernd Ludwig. *Ein konfigurierbares Dialogsystem für Mensch-Maschine-Interaktion in gesprochener Sprache*. Dissertation, Universität Erlangen-Nürnberg, 2004.

[Lud04b] Bernd Ludwig. A Pragmatics-First Approach to the Analysis and Generation of Dialogues. In Susanne Biundo, Rhom Frühwirth und Günther Palm, Hrsg., *Proc. KI-2004 (27th Annual German Conference on AI (KI-2004)*, Seiten 82–96, Berlin, 2004. Springer.

[Lud06] Bernd Ludwig. Tracing Actions Helps in Understanding Interactions. In *Proceedings of the SIGDIAL 06 Workshop on Discourse and Dialogue*, Seite to appear, 2006.

[MLSS06] Stefan Mandl, Bernd Ludwig, Sebastian Schmidt und Herbert Stoyan. Recurring Hidden Contexts in Online Concept Learning. In Adi Botea, Olivier Buffet und Marina Zanella, Hrsg., *Proceedings of the ECAI Workshop on Planning, Learning and Monitoring with Uncertainty and Dynamic Worlds*, Seite to appear, 2006.