Smarte Objekte und smarte Umgebungen

Programmierung durch den Benutzer
-
Programming by Demonstration

Lukas Stucki
Betreuung: Marc Langheinrich

28. 06. 2005

Abstract

In dieser Arbeit wird beschrieben, wieso der Benutzer überhaupt programmieren soll und welche generellen Ansätze es gibt, dies zu ermöglichen. Weiter wird auf „Programming by Demonstration“ eingegangen und es werden die Ziele und Schwierigkeiten erläutert. Am Schluss wird „a CAPpella“, ein Projekt das „Programming by Demonstration“ benutzt, näher vorgestellt und es werden Schlussfolgerungen gezogen.

1 Programmierung durch den Benutzer

1.1 Wieso soll der Benutzer programmieren?

Es ist für Entwickler schwierig, auf den Benutzer angepasste Programme zu schreiben. Könnte der Benutzer selber programmieren, so wäre er in der Lage, solche (kleine) Programme selber zu schreiben oder zumindest vorhandene Lösungen an seine Bedürfnisse anzupassen. Er müsste sich dann nicht mit Standardlösungen zu Frieden geben, sondern könnte sein individuelles Programm, angepasst an seine Bedürfnisse, selber schreiben oder zumindest bestehende Lösungen an seine Bedürfnisse anpassen.

Dies hätte auch den Vorteil, dass er bei Änderungen in seinem System das Programm selbständig anpassen könnte, ohne dass die Hilfe von professionellen Entwicklern benötigt würde, womit Kosten gespart werden könnten.

1.2 Mögliche Ansätze

Wichtig bei all diesen Ansätzen ist es, dass der Benutzer die „Programmierumgebung“ ohne grossen Lernaufwand und Vorkenntnisse effektiv nutzen kann. Ist dies nicht möglich, wird der Benutzer das System nach kurzer Zeit nicht mehr benutzen und der Nutzen bleibt aus.

Eine oft verwendete Art von Benutzer-Programmierung ist das Aufzeichnen von *Makros*. Der Benutzer kann das System ohne Vorkenntnisse verwenden und so wiederholende Aufgaben erledigen lassen. Das Problem mit den Makros ist die oft fehlende Verallgemeinerung, so

dass nur exakte Wiederholungen gemacht werden können.

Ein anderer Ansatz ist die *visuelle Programmierung*. Der Benutzer kann durch Zusammensetzen von gegebenen „Programmier-Bausteinen“ sein eigenes Programm schreiben. Der Benutzer muss keinen Syntax lernen. Er programmiert durch das Zusammenhängen von Bausteinen. Beispiele sind die Programmierumgebung zum Lego Mindstorm oder LabVIEW, ein Umgebung zur grafischen Erstellung von Messprogrammen.

Natürliche Programmiersprachen [1] verfolgen das Ziel, dass Benutzer die Programme in derselben Weise schreiben können, in der sie über das Problem denken. Dabei ist die ganze Programmierumgebung von grosser Bedeutung.

Skriptsprachen und *Programmspezifische Sprachen* versuchen, durch eine eingeschränkte Anzahl von Schlüsselwörtern, für den Benutzer einfach verwendbar zu sein. Die Kehrseite ist ein auf diese Schlüsselwörter eingeschränktes Anwendungsgebiet.

Es gibt weitere Ansätze, wie zum Beispiel „Programming by Demonstration“. Dieses Verfahren wird im nächsten Abschnitt ausführlicher behandelt.

2 Programming by Demonstration

2.1 Einleitung

Wäre es nicht toll, wenn man dem Computer einen Arbeitsablauf einmal zeigen könnte und der Computer das ganze, bei Erkennung ähnlicher Situationen, auf die Situation angepasst wiederholen würde? Der Computer nutzt das Beispiel, das durch den Benutzer vorgegeben wurde und wendet das Gelernte auf neue Situationen an.

Ein Beispiel ist der mechanische Wecker. Der Benutzer gibt mit dem roten Alarmzeiger ein Beispiel vor. Das System läuft nun, bis es das Gelernte wiedererkennt und führt dann das vorgegebene Programm aus, es klingelt.

Abbildung 1: mech. Wecker

2.2 Mögliche Anwendungsgebiete

Nach Allen Cypher [2] gibt es drei Anwendungsgebiete, die kurz mit Beispielen vorstellt werden:

Anpassungen von Programmen: Das „System“ wird für das automatische Beantworten von Dialogen, die immer gleich beantwortet werden, benutzt.

Automatisieren von wiederholenden Aufgaben: Mit den Fotos eines Verzeichnisses wird automatisch ein Web-Album generiert.

Schreiben von kleinen Programmen: Für ein Menüvorschlag-Programm werden die Daten des smarten Kühlschranks mit dem elektronischen Kochbuch kombiniert.

2.3 Ziel und Schwierigkeiten

Ziel von „Programming by Demonstration“ ist es, die Handlungen des Benutzers auf ähnliche Situationen anzuwenden. Dabei können Schwierigkeiten auftreten. Die Handlungen des Benutzers müssen korrekt interpretiert werden, d.h. das Wesentliche der Handlung muss

erkannt werden. Dies ist dann die Grundlage für eine korrekte Verallgemeinerung. Oft ist es recht schwieig die wesentlichen Teile einer Handlung zu bestimmen und diese dann richtig zu verallgemeinern.

Stellen wir uns vor, dass wir ein Bild bearbeiten möchten. Wir wählen einen Bereich in der oberen linken Ecke aus. Dies kann nun verschiedene Bedeutungen haben:

- wir möchten einen Bereich, der 100 x 200 Pixel gross ist
- wir wählen einfach die halbe Breite und Höhe
- wir möchten ein bestimmtes Objekt auf dem Bild auswählen
- ...

Um die Handlungen des Benutzers korrekt interpretieren zu können, muss das System die Entscheidungsgrundlage des Benutzers erkennen. Diese sind aber oft visuell oder im Kopf des Benutzers und deshalb für das System schwer erkennbar.

Es gibt aber Situationen, in denen es einfacher ist, etwas zu zeigen, als es zu beschreiben. Es ist beispielsweise enorm schwierig zu beschreiben, wie man die Schuhe bindet. Anderseits ist es relativ einfach, das zu zeigen. Allerdings setzt dies in vielen Fällen einen ähnlchen Erfahrungshorizont zwischen dem Zeigenden und dem Lernenden voraus.

3 „a CAPpella“

Dieses Kapitel bezieht sich im wesentlichen auf das Paper von Dey, et al. [4].

3.1 Übersicht

„a CAPpella“ steht für Context-Aware Prototping environment. Das System soll „programmierte“ Situationen, die von verschiedenen Sensoren registriert werden, erkennen und die zu dieser Situation gehörenden Handlungen ausführen.

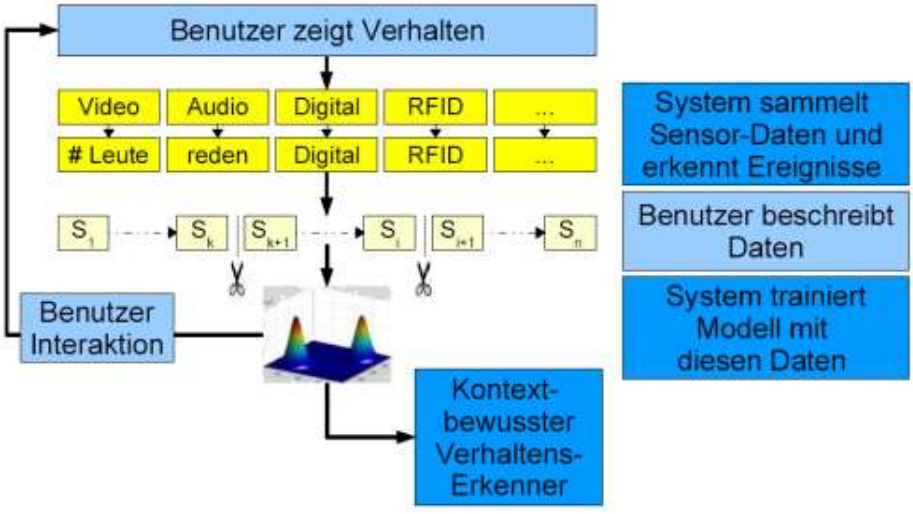

Abbildung 2: "a CAPpella" Benutzungsablauf. Tätigkeiten des Benutzers hellblau dargestellt

Es ist ein System für den Benutzer. Er soll es konfigurieren und einrichten können. Dazu wird „Programming by Demonstration“ eingesetzt. Der Benutzer zeigt dem System die Situationen, die es erkennen soll mitsamt den dazu gehörenden Handlungen, die es für den Benutzer ausführen soll. Der Benutzer kann also das System programmieren, ohne dass er Code eingeben muss. Das ist auch das Ziel, denn der Aufwand zum erlernen der Benutzung wie auch der Aufwand, der benötigt wird, um eine Situation zu „programmieren“, soll möglichst klein sein.

Weitere Anforderungen an das System sind breite Einsatzmöglichkeiten und dynamisches Anpassen an veränderte Situationen. Auf diesen Punkt wird zu einem späteren Zeitpunkt noch eingegangen.

3.2 Kontext-Bewusstsein

Es gibt grundsätzlich zwei Ansätze, wie das Problem des Kontext-Bewusstseins gelöst werden kann. Die eine ist regelbasiert. Wenn jemand zum Beispiel ein Programm zum Wässern des Gartens schreiben möchte, so kann er die Regel „Falls es seit zwei Tagen keinen Regen gegeben hat, dann wässern“ in eine dem Computer verständliche Form bringen. Wie aber bringt er ihm bei, dass bei einem Meeting das Licht anmacht werden soll?

Dazu kann der zweite, erkennungsbasierte Ansatz benutzt werden. Er verwendet, im Gegensatz zum ersten, künstliche Intelligenz, um Situationen zu erkennen. Die Schwierigkeit besteht darin, dass solche Systeme in der Regel schwierig zu konfigurieren sind und dass sie nicht einfach zu verstehen sind. Ob solche Systeme für den allgemeinen Benutzer geeignet sind, ist deshalb fraglich.

„a CAPpella“ verwendet einen erkennungsbasierten Ansatz mit dem Benutzer „in der Schleife“, so dass die Effizienz des Lern-Algorithmus verbessert wird.

Das Vorgehen wird im Folgenden am Beispiel eines Telefon-Meetings erläutert.

3.3 Telefon-Meeting

Würden wir zehn Leute nach einer Beschreibung von einem Meeting fragen, bekämen wir wahrscheinlich zehn verschiedene Antworten. Zeigen wir ihnen aber ein Bild, das diese Situation festhält, so werden die meisten wahrscheinlich derselben Meinung sein: die Situation stellt ein Meeting dar oder sie stellt kein Meeting dar. Diese Eigenschaft soll mit „a CAPpella“ ausgenutzt werden, indem „Programming by Demonstration“ mit einem erkennungsbasierten Ansatz benutzt wird.

Der Benutzer möchte das System zum Beispiel für eine Meeting-Situation nutzen. Man kann sich ein Telefon-Meeting vorstellen, bei dem der Benutzter zum Beispiel das Licht angeschaltet und ein Notiz-Programm gestartet haben möchte.

Falls der Benutzter bereit ist, das Kontext-Bewusste Verhalten zu erzeugen, startet er das Aufnahme-System von „a CAPpella“. Das nimmt die Daten von allen Sensoren auf. Die Sensoren können zum Beispiel Video-Kameras, Mikrofon oder der Computer sein. Das Meeting wird begonnen und der Benutzer macht alle Handlungen, die er gerne seinem smarten System übergeben würde.

Wenn das Meeting fertig ist, stoppt der Benutzer die Aufzeichnung und öffnet die Benutzeroberfläche (Abbildung 3). Hier kann er die aufgezeichneten Daten anschauen und wiedergeben. Die Benutzeroberfläche ist in drei Teile aufgeteilt. Im Wiedergabe-Fenster können die Audio- und Videodaten wiedergegeben werden. Im Ereignis-Fenster sind die aufgrund der Sensordaten erkannten Ereignisse dargestellt. Das Aktion-Fenster zeigt die vorgenommenen Handlungen auf. Der Benutzer soll nun die Daten beschreiben: Er wählt die relevanten Datenströme und die Handlungen, die das System übernehmen soll aus und setzt die Start- und Endzeitpunkte der zu erkennenden Situation (Abbildung 3).

Wenn er mit beschreiben fertig ist, trainiert „a CAPpella“ das Modell mit diesen Daten. Um

die Erkennungs-Fähigkeit zu erhöhen, muss der ganze Vorgang ein paar mal wiederholt werden.

Ist das System nun so gut trainiert, dass es die Situation regelmässig erkennt, kann der Benutzer das System anweisen, kontinuierlich Daten zu sammeln und die „programmierte“ Situation zu erkennen. Falls es eine Situation erkennt, werden die vorgegebenen Handlungen (Licht anzünden, Notiz-Programm starten, etc.) ausgeführt.

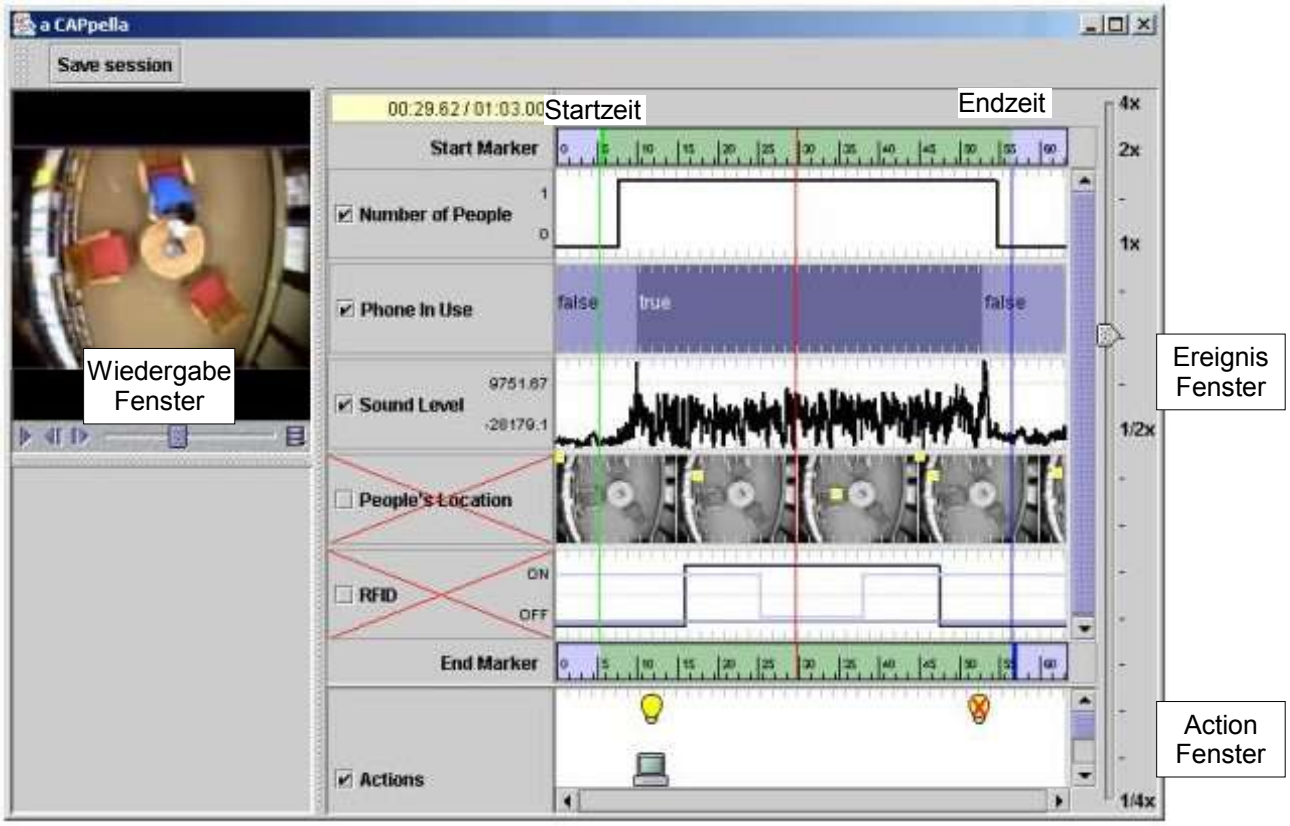

Abbildung 3: Benutzeroberfläche von "a CAPpella"

3.4 Das System und seine Komponenten

Das „a CAPpella“-System kann in vier Komponenten aufgeteilt werden: Aufnahme-System, Ereignis-Erkennung, Benutzeroberfläche und das Machine-Learning-System. Im Folgenden werden die einzelnen Komponenten kurz vorstellt.

Das *Aufnahme-System* zeichnet die Daten von den verschiedenen Sensoren gleichzeitig auf. Aufgenommen wird die Situation sowie die vorgenommenen Handlungen. Die in diesem Prototypen verwendeten Sensoren sind eine Overhead-Kamera, ein Mikrofon, RFID-Antennen und ein Telefon-Sensor, der aufzeigt, ob das Telefon benutzt wird oder nicht. Zum Aufzeichenen der Handlungen verwendet man einen instrumentalisierten Licht-Schalter und einen Computer, der die benutzten Programme, versendete Emails, etc. aufzeichnet. Grundsätzlich könnten weitere Sensoren hinzugefügt werden.

Die *Ereignis-Erkennung* wandelt die hochdimensionalen Daten in „höhere“ Ereignisse um. So werden dem Video die Anzahl Personen und deren Position entnommen. Aus den Audio-Daten werden die Lautstärke und ob ein Gespräch geführt wird extrahiert. Dazu werden, um diese Daten zu erhalten, weitere Machine Learning Algorithmen verwendet.

Die *Benutzeroberfläche* (siehe Abbildung 3) spielt eine wichtige Rolle im ganzen System. Die einfache und intuitive Benutzung steht im Vordergrund. Denn wird sie vom Benutzer nicht verstanden, kann sie falsch verwendet werden, wodurch die Qualität der Daten und somit auch des Resultats leidet. Deshalb wurde die Benutzeroberfläche in verschieden Entwicklugsstadien überprüft und angepasst. Ein wichtiger Aspekt ist die Darstellung der Sensordaten mit den verschieden Datentypen (siehe Ereignis-Fenster in Abbildung 3)

Das *Machine-Learning-System* ist für die Erkennung der Situationen verantwortlich. Verwendet wird ein „Dynamic Bayesian Network“, oder genauer ein „Hidden Markov Model“. Für jede Aktivität werden zwei Modelle trainiert, eines für die Aktivität und eines, wenn die Aktivität nicht stattfindet; also zum Beispiel das Meeting findet statt respektive es findet kein Meeting statt.

Die trainierten Modelle werden mit den aktuellen Daten der Sensoren verglichen und das Modell mit der höchsten Wahrscheinlichkeit wird ausgewählt. Verglichen wird periodisch (1 x pro Sekunde) mit einem Sliding Window von 10 Sekunden, was zu kurzen Verzögerungen führen kann. In diesem Punkt sehen die Entwickler noch ein Optimierungspotential.

Durch eine allgemeine Formulierung des Recognizers, kann dieser auch für verschiedene Situationen verwendet werden. Er wurde nicht auf eine spezielle Aufgabe, wie zum Beispiel Gesichtserkennung optimiert und ist deshalb flexibel, jedoch auch weniger spezifisch, einsetzbar. Er kann sich aber auch einer Veränderung der Situation anpassen, indem das Modell mit neuen Daten trainiert wird.

3.5 Fallstudie der Entwickler

Um die Brauchbarkeit und die Funktionstüchtigkeit des Systems zu zeigen, haben die Entwickler eine Fallstudie gemacht. Sie untersuchten die Erkennungsfähigkeit des Systems für die Situationen des Telefon-Meetings mit einer Person und für normale Meetings mit 2 Personen. Dazu hatten sie mehrere Szenen vorbereitet, in denen ein solches Meeting abgehalten wurde, sowie Szenen, in denen kein Meeting stattfindet. Sie wählten zufällig je 15 Meeting-Szenen aus, die sie dann eine nach der anderen bearbeiteten und dem System zum trainieren der Modelle gaben. Nach jeder Stufe wurden die Modelle mit anderen Szenen getestet, was zu den Resultaten in Abbildung 4 führte. Aus dieser Abbildung ist auch ersichtlich, dass für ein gutes Ergebnis fünf bis sechs Trainigsdurchgänge notwendig sind.

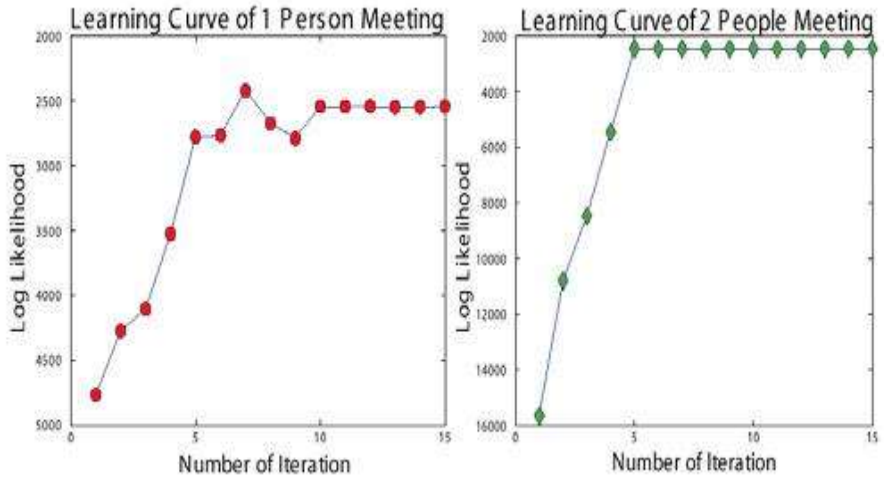

Abbildung 4: Lernkurve der Meeting-Szenario-Modelle

Nach der Trainingsphase (mit 15 Durchgängen) wurden die Modelle gegen die Test-Daten getestet. Die Test-Daten bestehen aus je 15 Szenen von Meetings mit einer Person am

Telefon und von Meetings mit zwei Personen und mit je 15 Szenen mit einer und zwei Personen, die kein Meeting hatten. Dabei kamen die guten Resultate aus Tabelle 1 zustande. So wurde zum Beispiel das Telefon-Meeting mit einer Person mit 93.3% erkannt, während das schlechteste Resultat, das Meeting mit zwei Personen doch noch zu 80.0% erkannt wurde.

	1P M	1P NM	2P M	2P NM
1P M	93.30%	6.60%	0.00%	0.00%
1P NM	13.30%	86.60%	0.00%	0.00%
2P M	0.00%	0.00%	80.00%	20.00%
2P NM	0.00%	6.60%	6.60%	86.60%

Tabelle 1: Erkennungmatirix nach 15 Trainingsdurchgänge für die Meeting-Szenarien (P=Person, M=Meeting, NM=nonmeeting)

3.6 Benutzer-Studie

Die grundsätzliche Brauchbarkeit von „a CAPpella“ wurde in der Fallstudie von den Entwicklern gezeigt. Doch das System ist nicht für die Entwickler, sondern für den Benutzer gedacht. Es muss also noch die Benutzbarkeit des Systems gezeigt werden. Zu diesem Zweck wurden 14 Probanden zwischen 18 und 60 Jahren, von denen keiner Informatiker ist, ausgesucht. Sie wurden fünf Minuten in die System-Benutzung eingeführt.

Getestet wurde, ob sie die Daten richtig bearbeiten können. Sie bekamen je drei Aufzeichnungen von Telefon-Meetings (mit einer Person) und von Meetings mit zwei Personen. Bei allen Daten-Sets mussten sie die relevanten Kanäle (Anzahl Personen, Lautstärke etc.) auswählen und die Start- und die Endzeit bestimmen. Mit den bearbeiteten Daten wurden dann die Modelle trainiert. Anschliessend wurden die Modell mit 60 Testszenen (4 x 15 Szenen) getestet. Dabei kamen folgende Resultate heraus: Szenen mit einer Person wurden durchschnittlich zu 67.2% richtig erkannt. Die persönlichen Resultate liegen zwischen 59.5% und 73.3%. Szenen mit zwei Personen wurden durchschnittlich nur mit 55.5% richtig erkannt, mit einer Streuung von 50.0% bis 78.6%.

Die Resultate sind, auch im Vergleich zu der Fallstudie der Entwickler, schlecht. Die Gründe nach Dey, et al. [4] sind die kleine Anzahl von Trainingsdurchgängen (in der Fallstude sieht man, dass 5-6 Trainingsdurchgänge für ein gutes Resultat notwendig sind. In der Benutzer-Studie hat man nur drei Trainingsdurchgänge gemacht.) und die vereinfachte Benutzeroberfläche. Die Entwickler verwendeten eine Oberfläche, bei der für jeden Kanal individuell mehrere Start- und Endzeitpunkte gewählt werden konnten. Die Benutzer konnten jeweils nur einen, für alle Kanäle gemeinsamen, Start- und Endzeitpunkt wählen.

4 Fazit

Das Resultat der Benutzer-Studie ist schlecht. Es kann jetzt, wie die Entwickler vermuten, an der Anzahl Trainingsdurchgängen liegen. Muss diese aber viel höher sein um gute Resultate zu erzielen, wird das Konfigurieren des Systems aufwendig und deshalb auch weniger benutzerfreundlich. Ein anderer Grund könnte das schlechte Verständnis der Benutzeroberfläche sein, die zu schlecht bearbeiteten Daten führte. Das könnte auch die grosse Differenz zwischen den verschiedenen Benutzern erklären.

Man könnte auch weitergehen und behaupten, dass sich „Programming by Demonstration“ für dieses Problem nicht eignet. Ich würde zwar nicht soweit gehen, doch stellt sich die Frage, wie kompliziert die zu bearbeiteten Probleme sein dürfen. Wie in Abschnitt 2.3 erläutert wurde, ist das Erkennen des Ziels einer Handlungen und die Verallgemeinerung davon schwierig. Es ist nicht einfach die für einen Vorgang typischen Handlungen

festzulegen. Je komplizierter das Problem wird, um so mehr mögliche Lösungen gibt es.

Dennoch würde ich „Programming by Demonstration“ als eine gute „Programmier-Möglichkeit“ für Benutzer bezeichnen. Der Programmierstil ist natürlich und intuitiv.

Literaturverzeichnis

[1] Brad A. Myers, John F. Pane, Andy Ko
Natural Programming Languages and Environments.
ACM, September 2004

[2] Allen Cypher (Ed.)
Watch What I Do – Programming by Demonstration.
MIT Press, Cambridge, MA, USA, 1993

[3] Henry Lieberman (Ed.)
Your Wish is my Command: Programming by Example.
Academic Press, 2001

[4] Anind K. Dey, Raffay Hamid, Chris Beckmann, Ian Li, Daniel Hsu
a CAPpella: Programming by Demonstration of Context-Aware Applications.
Proceedings of CHI 2004, Vienna, Austria, April 2004