DialogOS 1.1 Handbuch

CLT Sprachtechnologie GmbH

Inhaltsverzeichnis

Abbildungsverzeichnis

Kapitel 1

Einführung

Das Dialog-Management-Tool DialogOS ist eine Entwicklungsumgebung zum Erstellen, Testen und Ausführen von Sprachdialogsystemen. Mit DialogOS können Sie innerhalb kürzester Zeit technische Geräte per Sprache bedienen. DialogOS verfügt über eingebaute Komponenten zur Sprach- und Eingabe wie auch zur Steuerung eines LEGO Mindstorm NXT-Roboters. Das System kann außerdem über eine Modul-Schnittstelle mit beliebigen externen Modulen kommunizieren. Dies können z.B. Module sein, die mit dem Benutzer interagieren (weitere Spracherkenner, Sprachausgabe, Display etc.), aber auch externe Datenquellen (SQL-Datenbanken etc.) oder zu steuernde Geräte.

Achtung: Aufgrund der Komplexität natürlicher Sprache ist nach dem heutigen Stand der Technik eine 100% fehlerfreie Erkennung von Sprache nicht möglich. Verwenden Sie DialogOS daher auf keinen Fall in Umgebungen, in denen durch fehlerhafte Spracherkennung persönlicher oder materieller Schaden entstehen kann!

1.1 DialogOS starten

DialogOS speichert Dialogabläufe in sogenannten *Dialogmodellen*, die den Ablauf des Dialogs beschreiben. Sie starten DialogOS am einfachsten über den Eintrag im Windows Startmenü. Alternativ können Sie einen Doppelklick auf die Datei `DialogOS.exe` machen.

Wenn Sie DialogOS starten, öffnet sich zunächst der Startbildschirm. Sie können nun ein neues Dialogmodell erstellen oder ein bestehendes Dialogmodell öffnen.

Wenn Sie direkt im DialogOS-Verzeichnis ein Dialogmodell unter dem Namen `Model.xml` speichern, wird dieses beim Start von DialogOS automatisch geladen. Alternativ können Sie den Namen des zu öffnenden Modells beim Starten von DialogOS als Parameter mit der Option `-model` angeben, z.B.

```
DialogOS.exe -model MeinDialog.xml
```

Wenn Sie zusätzlich die Option `-execute` angeben, wird das Dialogmodell automatisch ausgeführt:

```
DialogOS.exe -execute -model MeinDialog.xml
```

Abbildung 1.1: Der DialogOS Startbildschirm

1.2 Aufbau eines Dialogmodells

Ein Dialogmodell besteht aus einer Reihe von *Dialogzuständen*, wobei jeder Dialogzustand eine bestimmte Funktion übernimmt, z.B. die Erkennung einer Benutzereingabe, die sprachliche Ausgabe einer Antwort des Systems oder das Senden eines Kommandos an eine zu steuernde Maschine. Durch Kombinieren dieser verschiedenen Zustände erstellen Sie ein System, das intelligent auf Ihre Antworten reagiert.

In DialogOS beschreiben Sie einen Dialog grafisch durch ein *Ablaufdiagramm*, das die verschiedenen Dialogzustände und die Übergänge von einem Zustand zum nächsten beschreibt. Man spricht in diesem Zusammenhang von einem *Graphen*, der aus verschiedenen *Knoten* besteht, die durch *Kanten* miteinander verbunden sind. Der Ablauf eines Dialogs beginnt immer in einem *Start*-Knoten (jeder Dialog hat immer genau einen Startknoten). Über die Kante wechselt der Dialog zum nächsten Knoten und führt diesen aus, d.h. die zu diesem Dialogzustand gehörende Funktion wird ausgeführt. Danach folgt der Dialog der ausgehenden Kante zum nächsten Knoten.

Von einem Knoten können auch mehrere Kanten ausgehen - dies entspricht einer Verzweigung im Dialog. Welche Kante auswählt wird, um zum nächsten Knoten zu wechseln, ergibt sich erst bei der Ausführung des Dialogs.

1.3 Das Arbeitsfenster

Wenn Sie ein neues Dialogmodell erstellen, erhalten Sie die Arbeitsfläche für einen neuen, leeren Dialog. Die Arbeitsfläche besteht aus vier Teilen.

Die Arbeitsfläche

Den größten Teil nimmt in der Mitte des Fensters die Arbeitsfläche für die Beschreibung des Dialogs ein. Zu Beginn enthält Ihr Dialog genau einen Startknoten. Da jeder Dialog immer genau einen Startknoten enthält, können Sie diesen weder löschen noch neue Startknoten hinzufügen.

Die Knotenleiste

Am rechten Rand des Bildschirms sehen Sie eine Leiste mit allen verfügbaren Knotentypen. Zur Erinnerung: Jeder Dialogzustand erfüllt eine bestimmte Funktion, die durch einen eigenen Knotentyp dargestellt wird. So gibt es Knoten für die Spracheingabe, die Sprachausgabe, die Steuerung eines angeschlossenen Lego Mindstorms Roboters, und viele andere Funktionen.

Symbolleiste

Am oberen Rand des Fensters sehen Sie die Symbolleiste, die Ihnen Zugriff auf häufig benötigte Funktionen in DialogOS bietet.

Prozedurliste

Um die Übersichtlichkeit zu erhöhen, können Sie mehrere Knoten zu einem eigenen Teildialog (man sagt auch Untergraphen) zusammenfassen. Am linken Rand des Fensters wird eine Liste aller Untergraphen in Ihrem Dialog dargestellt.

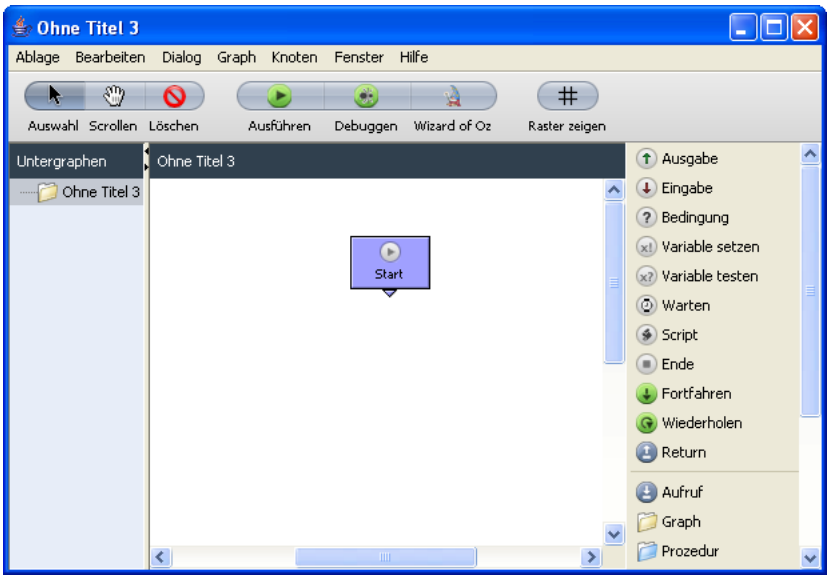

Abbildung 1.2: Die DialogOS Arbeitsfläche

Kapitel 2

Erste Schritte

Dieses Kapitel beschreibt die ersten Schritte mit DialogOS. Erzeugen Sie zunächst, wie im vorigen Kapitel beschrieben, einen neuen leeren Dialog.

2.1 Knoten und Kanten

Ein Dialogmodell besteht aus einer Liste von Knoten, die durch Kanten miteinander verbunden sind. Sie bearbeiten einen Dialog, indem Sie neue Knoten hinzufügen, diese konfigurieren und durch Kanten miteinander verbinden.

2.1.1 Hinzufügen von neuen Knoten

Um einen neuen Knoten zu einem Dialog hinzuzufügen, ziehen Sie diesen einfach per Drag&Drop aus der Knotenleiste am rechten Fensterrand in den Arbeitsbereich.

1. Klicken Sie mit der Maus in die Knotenleiste am rechten Fensterrand.
2. Ziehen Sie den Knoten bei gedrückter Maustaste in die Arbeitsfläche.
3. Lassen Sie die Maustaste los. An der Stelle, wo sich der Mauszeiger befindet, wird ein neuer Knoten des ausgewählten Typs eingefügt.

Ziehen Sie nun wie oben beschrieben einen *Ende*-Knoten auf den Arbeitsbereich und platzieren Sie ihn unterhalb des existierenden Startknotens.

2.1.2 Knoten verschieben

Sie können Knoten innerhalb des Arbeitsbereichs jederzeit verschieben, indem Sie mit der Maus auf den Knoten klicken und den Knoten bei gedrückter Maustaste an die gewünschte Stelle schieben.

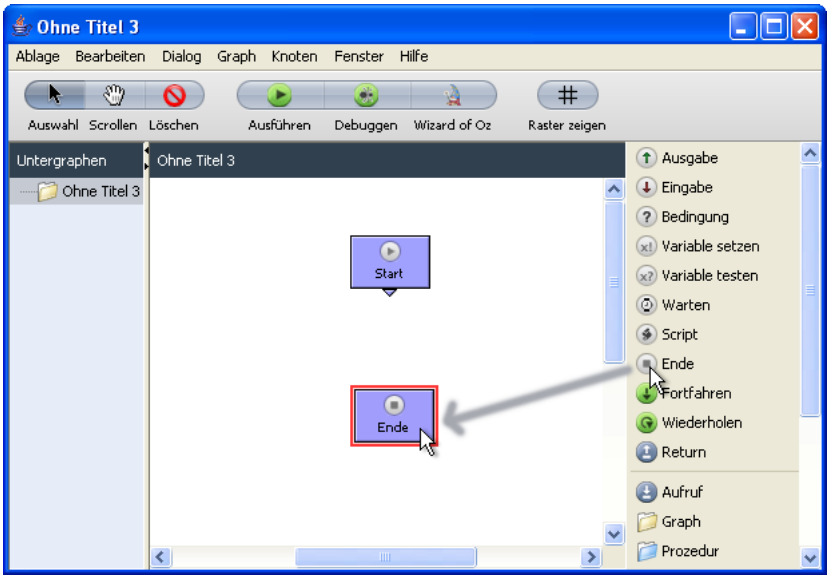

Abbildung 2.1: Einen neuen Knoten hinzufügen

2.1.3 Knoten konfigurieren

Jeder Knoten symbolisiert eine bestimmte Funktion innerhalb des Dialogs. Jede dieser Funktionen hat bestimmte Eigenschaften, die Sie für jeden Knoten individuell anpassen können. Um die Einstellungen für einen Knoten zu bearbeiten, machen Sie mit der Maus einen Doppelklick auf den Knoten. Daraufhin öffnet sich ein Dialogfenster in dem Sie die Eigenschaften des Knotens ändern können.

Abbildung 2.2: Die Eigenschaften eines Knotens

Wenn Sie den Dialog durch Klicken auf die Schaltfläche `OK` schließen, werden Ihre Änderungen übernommen. Wenn Sie stattdessen auf `Abbrechen` klicken, bleibt der Knoten unverändert.

Die Einstellungsmöglichkeiten sind für jeden Knotentyp unterschiedlich und werden innerhalb des Einstellungsfensters in mehreren Karteikarten untergliedert. Die erste Karteikarte `Allgemein` ist jedoch für alle Knoten gleich. Sie können dem Knoten einen Namen geben, seine Farbe ändern und einen Kommentar eingeben, um Details zum Dialogablauf zu notieren.

2.1.4 Knoten durch Kanten verbinden

Der Ablauf des Dialogs, d.h. die Reihenfolge, in der die verschiedenen Knoten des Dialogs ausgeführt werden, wird durch die Kanten bestimmt, die die Knoten miteinander verbinden. Jeder Knoten hat ein oder mehrere *Ausgänge*, von denen aus Kanten den Knoten verlassen. Diese werden durch kleine Dreiecke an der Unterseite des Knotens symbolisiert. Um zwei Knoten miteinander zu verbinden, müssen Sie eine Kante von einem Ausgang eines Knotens zu einem anderen Knoten ziehen.

1. Klicken Sie mit der Maus auf den Ausgang des Startknotens.
2. Bewegen Sie die Maus bei gedrückter Maustaste. Eine rot hervorgehobene Linie erscheint und folgt Ihrem Mauszeiger.
3. Bewegen Sie die Maus über den Endeknoten, den Sie nach obiger Beschreibung eingefügt haben. Der Knoten wird blau hervorgehoben,
4. Lassen Sie nun die Maustaste los. Damit haben Sie eine Kante vom Ausgang des Startknotens zum Endeknoten erzeugt.

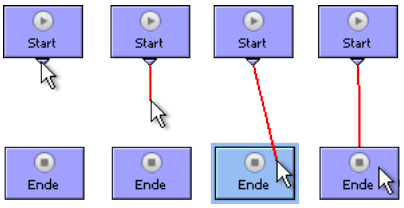

Abbildung 2.3: Zwei Knoten durch eine Kante verbinden

Sicherlich haben Sie bemerkt, dass der Endeknoten keinen Ausgang enthält. Sie können also den Ende-Knoten nicht mit weiteren Knoten verbinden. Das liegt daran, dass die Funktion des Endeknotens eben gerade ist, die Ausführung des Dialogs zu beenden.

2.1.5 Knoten und Kanten löschen

Um einen Knoten oder eine Kante zu löschen, wählen Sie diese zunächst aus, indem Sie mit der Maus auf den Knoten bzw. die Kante klicken. Diese sollte nun rot hervorgehoben werden. Wenn Sie die `Shift`-Taste dabei gedrückt halten, können Sie auch mehrere Knoten und Kanten gleichzeitig auswählen. Drücken Sie nun die `Entfernen`- oder die `Rückschritt`-Taste auf der Tastatur, um die ausgewählten Elemente zu löschen.

2.2 Den Dialog ausführen

Sie haben nun einen ersten Dialog erstellt, der aus einem Start- und einem Endknoten besteht. Sie haben diese durch eine Kante verbunden. Diesen einfachen Dialog können Sie nun ausführen. Wählen Sie dazu den Befehl `Ausführen` aus dem Menü `Dialog`, oder klicken Sie auf die Schaltfläche `Ausführen` in der Symbolleiste am oberen Bildschirmrand.

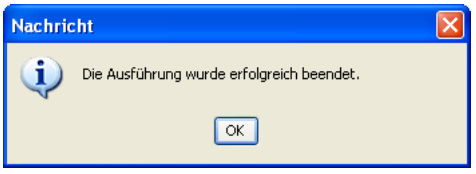

Abbildung 2.4: Erfolgreiche Ausführung eines Dialogs

Der Bildschirm blinkt kurz auf, danach erscheint eine Nachricht, dass der Dialog erfolgreich beendet wurde. Da Start- und Endeknoten jedoch keine eigene Funktion ausführen, sondern nur die Ein- und Ausstiegspunkte eines Dialogs markieren, ist nichts weiter passiert.

Aber die Nachricht weist immerhin darauf hin, dass der Dialog wie beabsichtigt abgearbeitet wurde. Dies ist jedoch nicht immer der Fall. So wird er Dialog z.B. mit einer Fehlermeldung abgebrochen, wenn Sie vergessen haben, den Ausgang eines Knotens durch eine Kante mit einem anderen Knoten zu verbinden. Bei der Ausführung weiß DialogOS dann nicht, mit welchem Knoten die Ausführung fortgesetzt werden soll.

Löschen Sie nun wie oben beschrieben die Kante zwischen dem Start- und dem Endknoten, und führen Sie den Dialog erneut aus. Sie erhalten nun die Meldung, dass die Ausführung mit einem Fehler unterbrochen wurde.

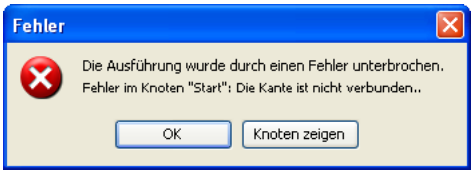

Abbildung 2.5: Fehler bei der Ausführung eines Dialogs

Wenn Sie auf die Schaltfläche `Knoten zeigen` klicken, zeigt Ihnen DialogOS automatisch den Knoten, an dem der Fehler aufgetreten ist, so dass Sie die Ursache des Fehlers beheben können.

2.3 Den Dialog abbrechen

Um einen laufenden Dialog abzubrechen, wählen Sie den Befehl `Abbrechen` aus dem Menü `Dialog`, oder klicken Sie auf die Schaltfläche `Abbrechen` in der Symbolleiste am oberen Bildschirmrand.

2.4 Den Dialog überprüfen

DialogOS bietet Ihnen die Möglichkeit, Fehler wie z.B. nicht verbundene Knoten schon vor der Ausführung zu finden. Damit können Sie vorab sicherstellen, dass Ihr Dialogmodell keine Probleme enthält, die bei der Ausführung zu einem Abbruch des Dialogs führen würden. Wählen Sie dazu im Menü `Graph` den Befehl `Überprüfen`. DialogOS überprüft nun Ihr Dialogmodell und zeigt Ihnen alle gefundenen Fehler an.

Sie haben nun gelernt, wie Sie neue Knoten zu einem Dialog hinzufügen, diese durch Kanten verbinden und den resultierenden Dialog überprüfen und dann ausführen können.

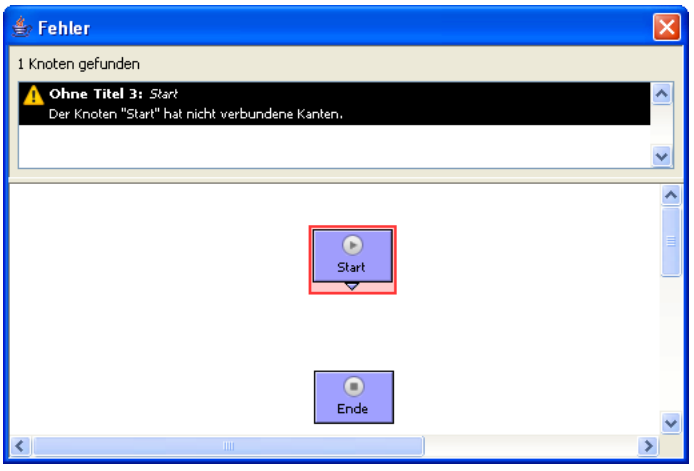

Abbildung 2.6: Fehler bei der Ausprüfung eines Dialogs

Kapitel 3

Spracherkennung und Sprachausgabe

Im vorigen Kapitel wurden die Grundlagen der Bedienung von DialogOS beschrieben. In diesem Kapitel lernen Sie nun, wie Sie die Komponenten zur Spracheingabe und Sprachausgabe nutzen, um Sprachdialoge zu entwerfen.

3.1 Sprachausgabe

DialogOS enthält ein eingebautes Text-to-Speech-System, das es erlaubt, beliebige Texte vorzulesen. Sie können also Ihren Computer sprechen lassen, einfach indem Sie den zu sprechenden Text eintippen. Voraussetzung ist natürlich, dass Ihr Computer eine Soundkarte hat und Sie einen Kopfhörer oder Lautsprecher daran angeschlossen haben. Für den Zweck der Sprachausgabe enthält DialogOS einen eigenen Knotentyp *Sprachausgabe*.

Erstellen Sie einen neuen Dialog und ziehen Sie einen *Sprachausgabe*-Knoten und einen Endknoten in die Arbeitsfläche und verbinden Sie diese mit Kanten, wie in der Abbildung zu sehen ist.

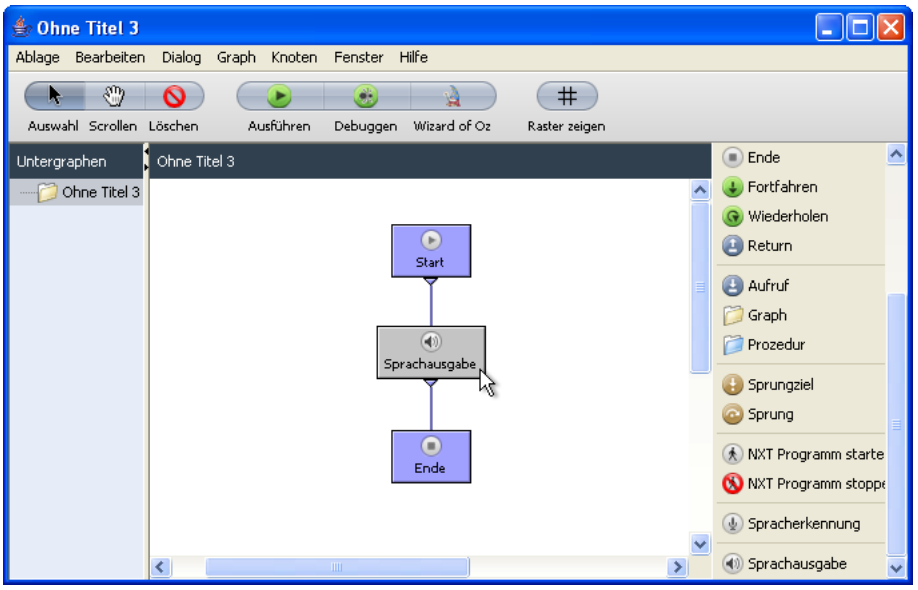

Abbildung 3.1: Sprachausgabe im Dialog

Machen Sie nun einen Doppelklick auf den Sprachausgabe-Knoten, um die Eigenschaften des Knotens zu bearbeiten. Klicken Sie auf den Karteireiter `Sprachausgabe`, um die Eigenschaften der Sprachausgabe zu konfigurieren.

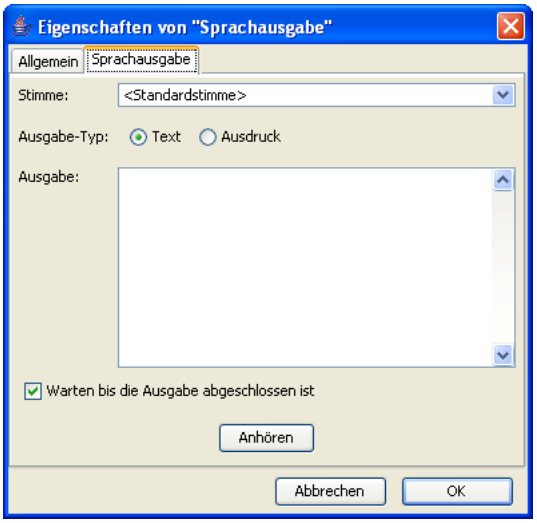

Abbildung 3.2: Eigenschaften der Sprachausgabe

An jedem Sprachausgabe-Knoten können Sie auswählen, mit welcher Stimme der angegebene Text gesprochen werden soll. Jede Stimme unterstützt dabei genau eine Sprache. Um z.B. einen deutschen Satz zu sprechen, müssen Sie also eine deutsche Stimme auswählen.

Im Feld `Ausgabe` geben Sie den Text ein, den das System sprechen soll. Sie können den zu sprechenden Text durch Auswahl des `Ausgabe-Typ` in zwei verschiedenen Formaten angeben: als einfachen Text oder als komplexen Ausdruck. Im Format `Text` wird der Text genau so gesprochen, wie Sie ihn eingeben. Dies ist das Standardformat. Im Format `Ausdruck` wird der Text als Formel interpretiert, dessen Ergebnis bei der Ausführung berechnet wird. Ausdrücke werden ausführlich in Kapitel 5.2 behandelt.

Über die Option `Warten bis die Ausgabe abgeschlossen ist` bestimmen Sie das Laufzeitverhalten der Sprachausgabe. Wenn die Option ausgewählt ist, wird der Dialog erst dann im nächsten Knoten fortgesetzt, wenn der Text vollständig gesprochen wurde. Dies ist das Standardverhalten. Wenn Sie die Option hingegen löschen, wird der Dialog sofort fortgesetzt, während das System noch spricht. So können Sie zum Beispiel die Spracherkennung aktivieren oder andere Dialogschritte ausführen, während gleichzeitig noch die Sprachausgabe läuft.

Durch Klick auf die Schaltfläche `Anhören` können Sie probeweise hören, wie das System Ihren Text mit der ausgewählten Stimme spricht.

3.1.1 Standardeinstellungen der Sprachausgabe

Wenn Sie komplexe Dialoge erstellen, die viele Sprachausgabe-Knoten verwenden, ist es mitunter aufwändig nd fehleranfällig an jedem Knoten die Stimme auswählen zu müssen. Aus diesem Grund haben Sie die Option, als Stimme die Option `<Standardstimme>` zu wählen.

Die Standardstimme für Ihren Dialog können Sie in den Dialogeinstellungen für die Sprachausgabe konfigurieren. Schließen Sie dazu das Eigenschaftenfenster des Knotens und wählen Sie im Menü `Dialog` den Befehl `Sprachausgabe`.

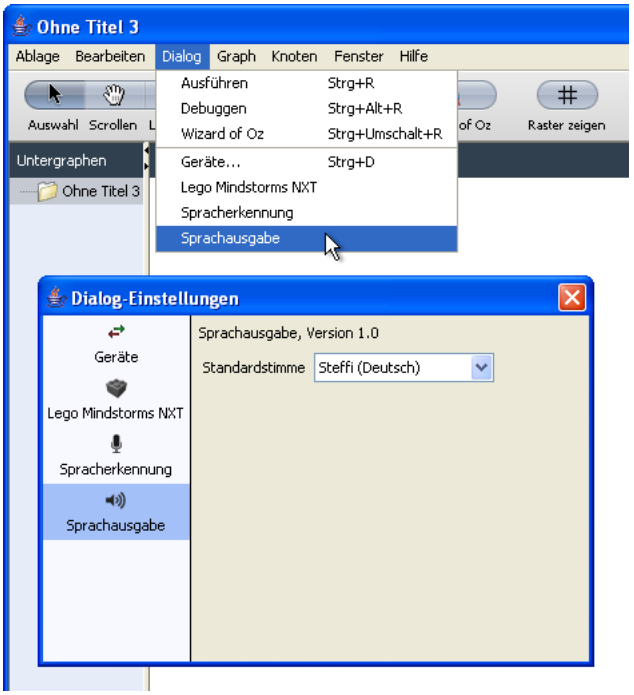

Abbildung 3.3: Dialogeinstellungen für die Sprachausgabe

Wählen Sie nun eine Stimme und geben Sie den zu sprechenden Text für den Sprachausgabe-Knoten ein. Wenn Sie den Dialog nun ausführen, spricht das System den Text.

3.2 Spracheingabe

DialogOS verfügt neben der Möglichkeit der Sprachausgabe auch über ein integriertes Modul zur Spracherkennung. Sie benötigen eine Soundkarte mit angeschlossenem Mikrofon, um die Spracheingabe nutzen zu können. Auch für die Spracheingabe gibt es in DialogOS einen eigenen Knotentyp.

Die Spracherkennung in DialogOS funktioniert sprecherunabhängig, d.h. DialogOS kann jeden beliebigen Sprecher erkennen, ohne auf dessen Stimme trainiert zu werden.[1] Allerdings müssen Sie DialogOS im Gegenzug beschreiben, was Sie in einer bestimmten Dialogsituation sagen können wollen. Denn natürliche Sprache ist so komplex, dass die Spracherkennung nicht jeden beliebigen Text automatisch verstehen kann. Stattdessen müssen Sie eine Liste von Worten oder Sätzen vorgeben, die für die Erkennung in Frage kommen. Erst durch diese Einschränkung des möglichen Vokabulars kann die Spracherkennung zuverlässig funktionieren.

Wenn Sie einen neuen *Spracheingabe*-Knoten in den Arbeitsbereich ziehen, hat dieser zunächst keinen Ausgang. Das liegt daran, dass Spracheingabe-Knoten im Gegensatz zu den Knotentypen, die Sie bisher kennen gelernt haben, keine feste Zahl von Ausgängen haben. Die Zahl der Ausgänge wird vielmehr geregelt durch die Zahl der unterschiedlichen Sprachkommandos, die Sie an diesem Knoten erkennen können wollen. Die Idee dabei ist, dass Sie für den Computer angeben, welche Kommandos er verstehen können soll. Für jedes Kommando erstellt DialogOS einen Ausgang am Spracheingabe-Knoten, so dass Sie für jedes Kommando eine Kante zu einem anderen Folgeknoten ziehen können. Aus diese Weise erhalten Sie eine Verzweigung im Dialogablauf, je nachdem welches Sprachkommando Sie dem Computer geben.

[1] Natürlich müssen Sie klar und deutlich sprechen. Ein starker Akzent oder Dialekt verschlechtert das Erkennungsergebnis.

Die Kommandos können Sie wie gewohnt bearbeiten, indem Sie einen Doppelklick auf den Spracheingabe-Knoten machen.

Abbildung 3.4: Einstellungen für die Spracheingabe

Jeder *Spracheingabe*-Knoten benötigt eine *Grammatik* und eine Liste von *Mustern*, die auf diese Grammatik passen. Die Grammatik beschreibt dabei für die Spracherkennung, welche Worte und Wortfolgen erlaubt sein sollen. Anhand der Muster kann DialogOS dann erkennen, welches Kommando Sie gesprochen haben.

Die Aufteilung in Grammatik und Muster dient vor allem der Beschreibung komplexer Kommandos und wird ausführlich in 8.2.1 beschrieben. Für einfache Kommandos bietet DialogOS die Möglichkeit, die Grammatiken automatisch für Sie aus den Mustern zu generieren. Wählen Sie dazu als Grammatik die Option `<Automatisch aus den Mustern generieren>`. Geben Sie nun die Sprache an, die Sie für die Kommandos verwenden wollen.

Nun brauchen Sie nur noch die Kommandos vorzugeben, die DialogOS erkennen können soll. Um ein Kommando hinzuzufügen, klicken Sie auf die Schaltfläche `Neu`. Um Kommandos zu löschen, klicken Sie auf das Kommando und dann auf die Schaltfläche `Löschen`.

Abbildung 3.4 zeigt ein einfaches Beispiel für einen Knoten, an dem DialogOS die Worte *ja* und *nein* verstehen und unterscheiden soll. Statt einzelnen Worten können Sie natürlich auch ganze Sätze als Kommando angeben. Beachten Sie jedoch, dass Sie bei der Ausführung des Dialogs das Kommando *genau so* sagen müssen, wie angegeben. Das heißt auch, dass z.B. *ja* und *ja bitte* unterschiedliche Kommandos sind.

Wenn Sie nun den Einstellungsdialog durch Klicken der Schaltfläche `OK` schließen, sehen Sie dass der Knoten nun zwei Ausgänge hat, einen für das Kommando *ja* und einen für das Kommando *nein*. Wenn Sie den Mauszeiger über einen Ausgang bewegen, erscheint nach kurzer Zeit ein Tooltipp, der das zu diesem Ausgang gehörige Kommando anzeigt.

Durch Klick auf die Schaltfläche `Ausprobieren` können Sie die Funktionsweise der Spracherkennung testen.

Wenn Sie den Dialog ausführen, wird die Spracherkennung aktiviert, sobald die Ausführung den *Spracheingabe*-Knoten erreicht. Sie sehen dies auch in der Benutzeroberfläche von DialogOS wie in Abbildung 3.5. Je nachdem, welches Kommando erkannt wurde, wird der Dialog dann über den entsprechenden Ausgang und die damit verbundene Kante fortgesetzt.

Abbildung 3.5: Aktive Spracheingabe

3.2.1 Standardeinstellungen für die Spracheingabe

Analog zur Sprachausgabe können Sie auch für die Spracheingabe die Standardsprache festlegen, damit Sie dies nicht an jedem Spracheingabe-Knoten erneut tun müssen. Wählen Sie dazu aus dem Menü `Dialog` den Befehl `Spracheingabe`.

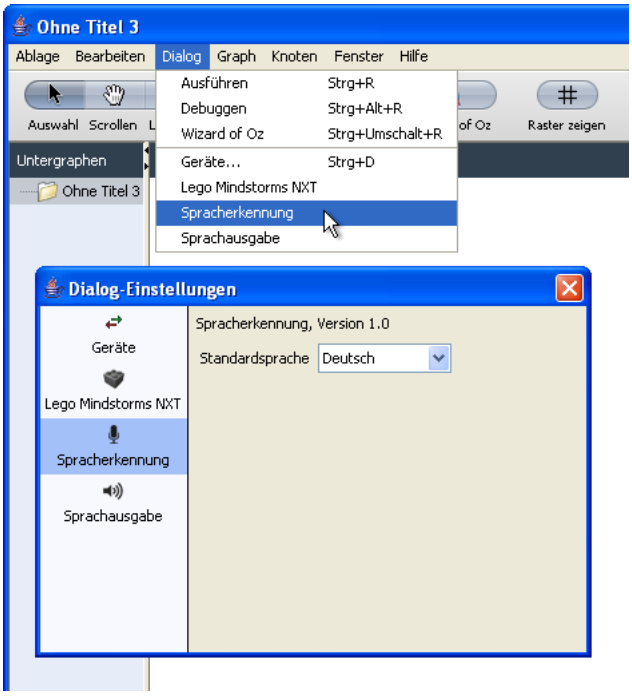

Abbildung 3.6: Einstellungen für die Spracheingabe

3.2.2 Weitere Optionen für die Spracheingabe

Das Konfigurationfenster für Spracheingabe-Knoten enthält noch einen weiteren Karteireiter `Optionen`.

Hier können Sie eine Zeitbegrenzung für die Spracheingabe vorgeben. Wenn Sie die Option aktivieren, können Sie in dem Textfeld eine Zeit in Millisekunden eingeben, z.B. *1000* für die Dauer von einer

Sekunde. Durch Aktivieren der Option erhält der Spracheingabe-Knoten einen zusätzlichen Ausgang. Wenn die Spracherkennung nach der vorgegebenen Zeit ab Aktivierung kein Sprachsignal erfassen konnte, wird die Spracherkennung abgebrochen und der Dialog über die separate Kante für die Zeitüberschreitung fortgesetzt.

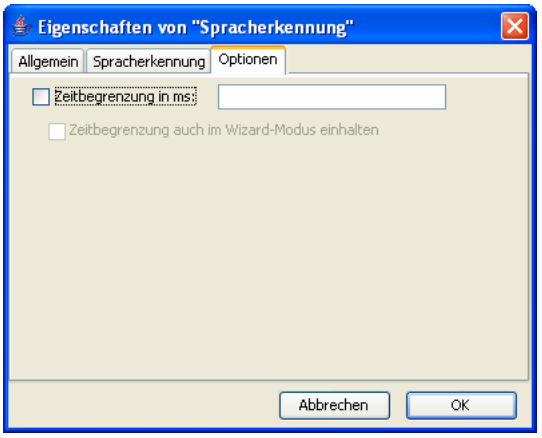

Abbildung 3.7: Weitere Optionen für die Spracheingabe

Kapitel 4

Lego Mindstorms Roboter steuern

In 3 haben Sie gelernt, wie Sie mit Spracheingabe- und Sprachausgabe-Knoten einen Dialog aufbauen können, in dem der Computer mit Ihnen spricht und Ihre gesprochenen Kommandos erkennt und in Abhängigkeit von Ihren Kommandos weiterführt. In diesem Kapitel wird beschrieben, wie Sie diese Sprachsteuerungsmöglichkeit nutzen können, um einen Roboter steuern zu können.

4.1 Auswahl des Roboters

Zunächst müssen Sie einstellen, welchen Roboter Sie steuern wollen. Öffnen Sie dazu die entsprechende Dialogeinstellung, indem Sie im Menü `Dialog` den Befehl `Lego Mindstorms NXT` auswählen.

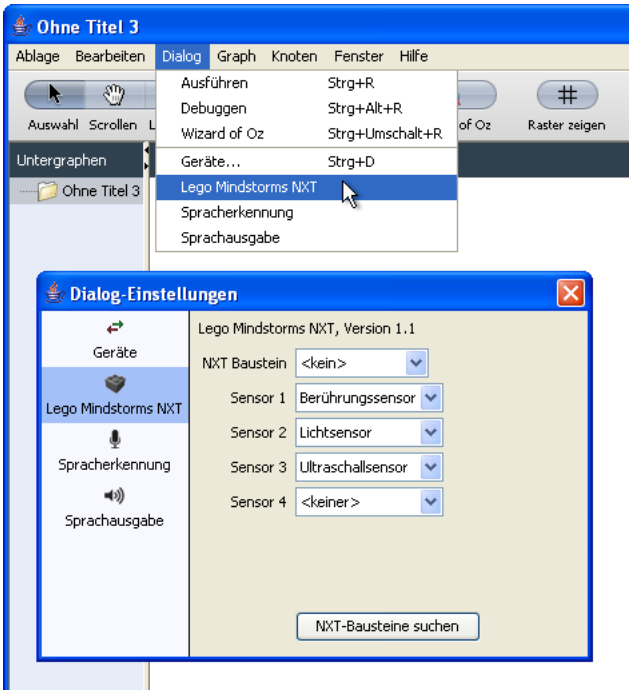

Abbildung 4.1: Dialogeinstellungen für Lego Mindstorms

Stellen Sie sicher, dass der NXT-Baustein eingeschaltet und erreichbar ist. Sie können den NXT per USB mit dem Computer verbinden, oder Bluetooth Funktechnik verwenden, falls Ihr Computer über

eine Bluetooth-Schnittstelle verfügt. Bitte stellen Sie außerdem sicher, dass Sie die neueste Version des Lego Mindstorms Treibers installiert haben und ihr NXT die aktuelle Firmware benutzt. Die Firmware können Sie ggf. mit der Mindstorms-Software aktualisieren, die Ihrem NXT beiliegt. Aktuelle Versionen des Treibers und der Firmware finden Sie unter `http://www.mindstorms.com/support/updates/`

Klicken Sie nun auf die Schaltfläche `NXT-Bausteine suchen`, damit DialogOS Ihren angeschlossenen NXT finden kann. Dieser sollte sich nun im Aufklappmenü `NXT Baustein` auswählen lassen.

Als nächstes können Sie angeben, welche Sensoren Sie an Ihren NXT angeschlossen haben. Damit haben Sie die Möglichkeit direkt von DialogOS aus aktuelle Sensorwerte abzufragen und im Dialog entsprechend darauf zu reagieren.

4.2 NXT Knoten

Zur Steuerung des Roboters stellt DialogOS drei Knotentypen bereit: *NXT Programm starten*, *NXT Programm stoppen* und *NXT Sensor abfragen*.

Sie können mit der Software, die Ihrem NXT beiliegt, mehrere Programme entwickeln und auf Ihren NXT laden. Wenn Sie z.B. den AlphaRex aus der Anleitung Ihres NXT gebaut haben, dann wissen Sie, wie Sie ein Programm entwerfen, dass die Motoren B und C startet, damit AlphaRex vorwärts läuft, und wie Sie ein Programm entwerfen, dass nur Motor A startet, damit AlphaRex die Arme bewegt. Speichern Sie diese beiden Programme unter dem Namen *Beine* bzw. *Arme* und laden Sie die auf den NXT.

DialogOS kann nun dem NXT jederzeit den Befehl schicken, eines dieser beiden Programme zu starten oder wieder zu stoppen.

4.2.1 Ein NXT Programm starten

Ziehen Sie zunächst einen Knoten vom Typ *NXT Programm starten* auf die Arbeitsfläche. Machen Sie nun einen Doppelklick auf den Knoten, um die Einstellungen zu bearbeiten.

Abbildung 4.2: Einstellungen für NXT Programm starten

Sie können den Namen des Programms angeben, dass auf dem NXT gestartet werden soll. Wenn Sie sich nicht mehr sicher sind, wie Sie das Programm genannt haben, können Sie auf die Schaltfläche `Auswählen` klicken, um eine Liste aller auf dem NXT geladenen Programme zu erhalten.

Bei der Ausführung des Dialogs sendet DialogOS beim Erreichen dieses Knotens nun einen Befehl an den NXT, das ausgewählte Programm zu starten.

Über die Option `Warten, bis das Programm zu Ende ist` können Sie bestimmen, wann DialogOS mit der Ausführung des nächsten Knotens fortfährt. Wenn Sie Option aktivieren, wartet DialogOS, bis der NXT meldet, dass das Programm abgeschlossen ist. Dies ist zum Beispiel sinnvoll, wenn Sie ein Programm geschrieben haben, mit dem sich der Roboter eine vordefinierte Strecke fortbewegt und der Dialog erst weitergehen soll, wenn der Roboter damit fertig ist. Wenn Sie die Option deaktivieren, fährt DialogOS sofort mit der Ausführung des Dialogs fort, noch während der Roboter sich bewegt.

Anmerkung: Wenn Sie ein NXT Programm geschrieben haben, dass selbst in einer Endlosschleife läuft, sollten Sie die Option unbedingt deaktivieren, da DialogOS sonst unendlich lange wartet. Insbesondere in diesem Fall ist es nützlich, ein Programm von DialogOS aus wieder abbrechen zu können.

Anmerkung: Auf dem NXT kann zu jederzeit nur *ein* Programm laufen. Sie können also nicht mehrere Knoten hintereinander verwenden, um mehrere Programme gleichzeitig auf dem NXT zu starten.

4.2.2 Ein NXT Programm abbrechen

Sie können ein auf dem NXT laufendes Programm jederzeit abbrechen. Ziehen Sie dazu einen Knoten vom Typ *NXT Programm stoppen* auf die Arbeitsfläche und machen Sie einen Doppelklick auf den Knoten, um die Einstellungen zu bearbeiten.

Abbildung 4.3: Einstellungen für NXT Programm stoppen

Im Normalfall hält die Ausführung dieses Knotens einfach das aktuell laufende Programm auf dem NXT an. Falls gar kein Programm auf dem NXT lief, passiert einfach gar nichts. Sie können jedoch die Option `Zuerst prüfen, ob ein Programm läuft` aktivieren, damit DialogOS unterscheidet, ob überhaupt ein Programmaktiv war. Der Knoten zum Stoppen des Programms erhält dann einen zweiten Ausgang, über den Sie den Dialog verzweigen können. Falls ein Programm lief, wird es angehalten und der Dialog über die erste Kante des Knotens fortgesetzt. Falls kein Programm lief, wird der Dialog über die zweite Kante fortgesetzt.

4.2.3 Sensorwerte abfragen

Sie können die an Ihren NXT angeschlossenen Sensoren auch direkt abfragen, um im Dialog darauf zu reagieren.

Sie müssen zunächst auswählen, welchen Sensoranschluss Sie abfragen wollen. In den Voreinstellungen müssen Sie dazu zunächst angeben, welche Art von Sensor Sie mit welchem Anschluss verbunden haben. Im Falle des Lichtsensors können Sie zusätzlich angeben, ob die eigene Lichtquelle des Sensors aktiviert werden soll.

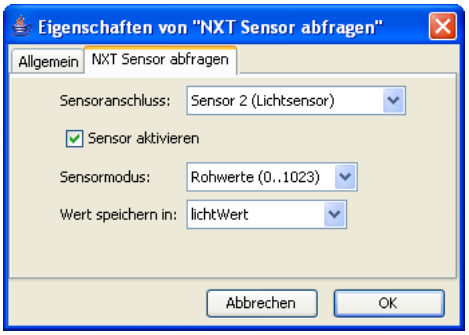

Abbildung 4.4: Einstellungen für NXT Sensor auslesen

Zusätzlich müssen Sie angeben, in welcher Einheit der NXT den Sensorwert liefern soll: als Rohwert, als boolschen Wert oder als Prozentangabe. Den ausgelesen Wert speichert DialogOS bei der Ausführung des Dialogs in einer Variablen Ihrer Wahl (für eine Einführung in Variablen siehe Kapitel 5.1).

4.3 Sprache und Robotersteuerung kombinieren

Sie können nun Ihren Roboter per Sprache steuern, indem Sie Knoten für Spracheingabe, Sprachausgabe und NXT-Steuerung per Kanten miteinander verbinden.

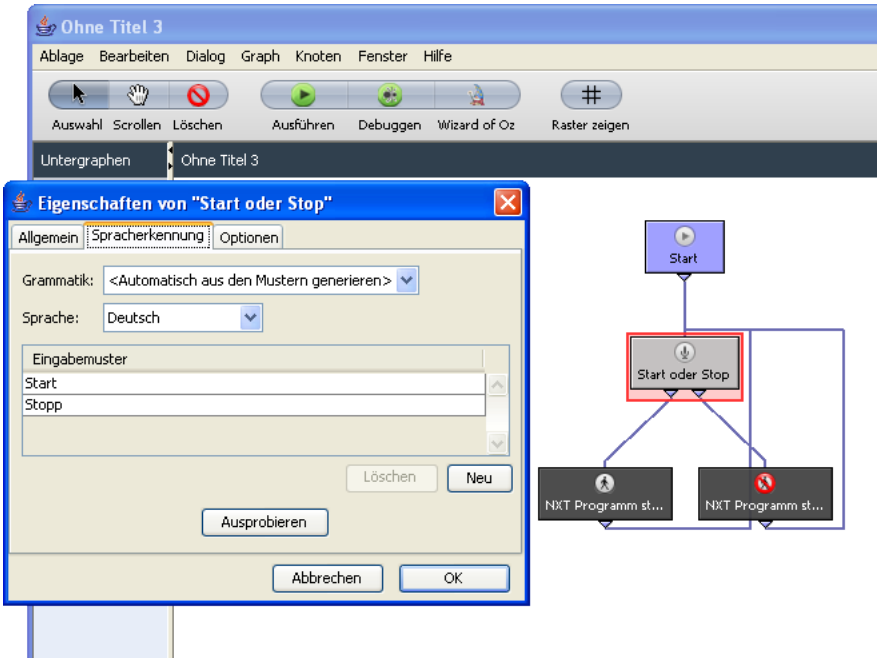

Abbildung 4.5: Ein einfacher Sprachdialog für Roboter

Abbildung 4.5 zeigt einen sehr einfachen Dialog, bei dem ein Programm auf dem NXT gestartet wird, wenn Sie *Start* sagen, und wieder angehalten wird, wenn Sie *Stopp* sagen.

Kapitel 5

Programmierung in DialogOS

Mit der Größe eines Dialogsystems steigt auch die Komplexität der möglichen Dialogabläufe. Um diese Komplexität zu bewältigen benötigt man häufig so etwas wie ein *Dialoggedächtnis*, d.h. die Möglichkeit, sich an einer Stelle im Dialog etwas zu merken und später darauf zurückgreifen zu können.

DialogOS bietet Ihnen so ein Gedächtnis in Form von *Variablen*. Der Begriff *Variable* ist dem Bereich der Programmiersprachen entliehen, und in der Tat bietet Ihnen DialogOS eine eigene eingebaute Programmiersprache, um beliebig komplexe Dialogfunktionalität zu implementieren. Diese Programmiersprache wird im folgenden Kapitel näher beschrieben.

5.1 Datentypen und Variablen

Eine Variable ist ein Speicherplatz, an dem Sie eine Information ablegen und später wieder abrufen können. Jede Variable hat einen *Datentyp*, der beschreibt, welche Art von Information in einer Variablen gespeichert werden kann, z.B. eine Zahl, oder ein Text. DialogOS unterstützt die folgenden Datentypen:

bool
: Mit dem Datentyp `bool` lassen sich die beiden Wahrheitswerte `true` (wahr) und `false` (falsch) darstellen. Jedes Ergebnis einer logischen Operation hat den Typ bool.

int
: `int` dient der Darstellung ganzzahliger Werte wie beispielsweise `1`, `312`, `-21`.

real
: `real` dient der Darstellung von realen Zahlenwerten mit dem `.` als Dezimaltrennzeichen wie beispielsweise `3.141`, `-2.8`, `5.0`.

string
: Mit dem Datentyp `string` lassen sich Zeichenketten darstellen. Der Wert eines Strings ist eine Zeichenkette, die am Anfang und am Ende mit einem doppelten Anführungszeichen gekennzeichnet wird wie beispielsweise `"DialogOS"`, `"Hallo"`, `""`.

list
: Mit dem Datentyp `list` lassen sich Listen von Daten darstellen, wobei der Datentyp eines jeden Listeneintrags beliebig ist. Es ist also insbesondere möglich Listen von Listen zu erstellen. Der Wert

einer Liste wird am Anfang und am Ende mit eckigen Klammern gekennzeichnet und Elementwerte mit Komma separiert wie beispielsweise `[0, 1, 2]`. Die leere Liste wird durch den Ausdruck `[ ]` repräsentiert. Die Elemente einer Liste müssen nicht alle den gleichen Typ haben.

struct
: Der Datentyp `struct` dient der Darstellung von Mengen von Daten, bei denen jedes Element eine Bezeichnung hat. Der Wert einer Struktur wird am Anfang und am Ende mit Mengenklammern gekennzeichnet und ihre Elemente mit Komma separiert. Jedes Element ist eine Wertzuweisung der Form "name=wert". Die leere Struktur wird durch den Ausdruck `{}` repräsentiert. Weitere Beispiele für Strukturen: `{ eins = 1, zwei = 2 }`, `{ zahlen = { eins=1 }, buchstabe = "a" }`

Jede Variable hat standardmäßig zunächst (unabhängig von ihrem Datentyp) den Wert `undefined`. Dieser Wert zeigt an, dass noch keine Information in der Variablen gespeichert wurde.

Bevor Sie eine Variable benutzen können, müssen Sie sie definieren. Um neue Variablen zu definieren oder existierende Variablendefinitionen zu bearbeiten, wählen Sie im Menü `Graph` den Befehl `Variablen`.

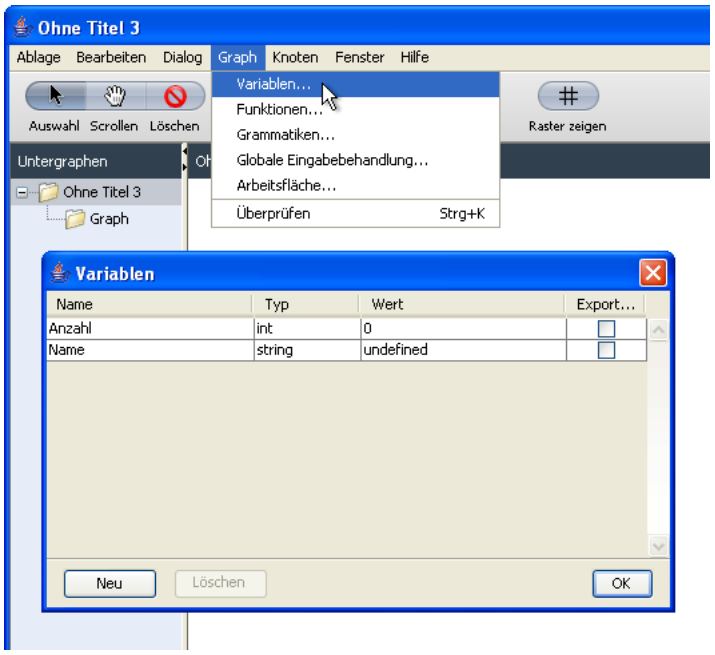

Abbildung 5.1: Definition von Dialogvariablen

Hier können Sie neue Variablen hinzufügen und bestehende Definitionen ändern oder löschen. Jede Variable hat einen Namen, einen Datentyp und einen initialen Wert. Statt eines konstanten Wertes können Sie für die Initialisierung auch einen komplexen Ausdruck (siehe Kapitel 5.2) angeben, der zur Laufzeit ausgewertet wird. Über die Option `Exportieren` können Sie angeben, ob diese Variable auch in Untergraphen sichtbar sein soll (Untergraphen werden ausführlich in Kapitel 10.6 beschrieben).

5.2 Ausdrücke

DialogOS bietet an vielen Stellen die Möglichkeit, Ausdrücke zu verwenden, um komplexe Gegebenheiten und Operationen darzustellen, beispielsweise in Ausgabeknoten und Bedingungen. In den folgenden Abschnitten wird gezeigt, welche Ausdrücke es gibt und welche Verwendungszwecke sie haben können.

5.2.1 Arithmetische und Boolsche Operatoren

Ausdrücke bestehen meist aus Operatoren und Variablen. Die Möglichkeiten der Verwendung von Operatoren sind unter anderem abhängig von den Datentypen (s. Kapitel 5.1) der verwendeten Variablen. Außerdem ergibt sich der Datentyp eines Ausdrucks wiederum aus den verwendeten Operatoren und Variablen.

Ein einfacher Ausdruck ist eine Variable oder ein (konstanter) Wert, der mit einem Datentyp dargestellt werden kann, z.B.:

```
meineNeueVariable
true
false
undefined
[undefined, 1, 3.1]
{ zahlen=[0,1,2,3], buchstaben= { latin=[a,b,c] } }
[[0,1], [1,0]]
```

Komplexe Ausdrücke bestehen aus Variablen oder konstanten Werten, die durch Operatoren miteinander verbunden sind. Ausdrücke können nach Belieben geklammert werden, um die Auswertungsfolge festzulegen. Werden Operationen nicht besonders geklammert, so sind sie im Allgemeinen linksassoziativ. Komplexe Ausdrücke können beliebig tief ineinander geschachtelt werden.

Für Ausdrücke, deren Wert eine Zahl ist, gibt es einen unären Operator `-`, der den Wert invertiert, und für Wahrheitswerte den Operator `!`, der aus `true` den Wert `false` macht und umgekehrt. Das Ergebnis dieser Operatoren hat den gleichen Typ wie ihre Operanden.

Weiterhin gibt es folgende binären Operatoren:

`+`

: Der Operator `+` dient bei der Verwendung von Zahlen als Addition. Dies gilt immer dann, wenn rechter und linker Operand Ausdrücke sind, die entweder zum Datentyp real oder zu int ausgewertet werden. Das Ergebnis hat den gleichen Datentyp wie die beiden Operand. Ferner ist es möglich, int- und real-Werte miteinander zu addieren, wobei das Ergebnis dann einen real-Wert liefert und string-Werte mit string-, int- oder real-Werten zu addieren, wobei das Ergebnis dann einen string-Wert liefert, indem die Zeichen des rechten Operanden dem linken Operanden rechts angefügt wurden.

`-`

: Der Operator `-` dient der Subtraktion von Zahlen. Er subtrahiert, analog zum Additionsoperator, int- und real-Werte. Bei der Verwendung anderer Datentypen kommt es zu einem Laufzeitfehler.

`*`

: Der Operator `*` dient der Multiplikation von Zahlen. Er multipliziert, analog zu den anderen arithmetischen Operatoren, int- und real-Werte und wirft einen "Type error"bei der Verwendung anderer Datentypen.

`/`

: `/` dividiert zwei Zahlen, analog zu den anderen arithmetischen Operatoren. Wenn der rechte Operand `0` ist, tritt ein Laufzeitfehler auf und die Ausführung des Dialogs wird abgebrochen.

`%`

: `%` liefert den Rest einer Ganzzahl-Division. In DialogOS funktioniert er neben int-Werten auch mit real-Werten, wobei der rechte Operand dann gerundet wird.

`::`

Der Operator `::` konkateniert ein Element beliebigen Typs mit einer Liste. Der linke Operand wird also einer Liste, die vom rechten Operanden bezeichnet wird, an vorderster Stelle angefügt. Die Operation ist rechtsassoziativ und liefert daher beispielsweise für den Ausdruck `x::y::[]` das Ergebnis `[x, y]`.

`<, >, <=, >=`

Die Operatoren `<`, `>`, `<=` und `>=` entsprechen den mathematischen Relationen mit den gleichen Symbolen. Beide Operanden müssen den gleichen Typ, `int` oder `real`, haben. Das Ergebnis dieser Operation ist immer vom Typ `bool`.

`==`

Der Operator `==` entspricht der äquivalenzrelation der Gleichheit. Die Operanden können jeden beliebigen Typ haben, unter der Voraussetzung, dass beide Operanden einen Wert des gleichen Typs ausdrücken. Der Operator liefert true, wenn beide Operanden den gleichen Wert besitzen, ansonsten false.

`!=`

Der Operator `!=` entspricht der Negation der Gleichheit. Dieser Operator funktioniert analog zu `==`, evaluiert aber zu false, wenn beide Operanden den gleichen Wert besitzen, ansonsten true.

`&&`

Der Operator `&&` entspricht der logischen Konjunktion. Mit ihm lassen sich zwei boolsche Ausdrücke miteinander verbinden, wobei `&&` genau dann `true` liefert, wenn beide Operanden zu `true` ausgewertet wurden, ansonsten `false`.

`||`

Der Operator `||` entspricht der logischen Disjunktion. Mit ihm lassen sich zwei boolsche Ausdrücke miteinander verbinden, wobei `||` genau dann `true` liefert, wenn mindestens einer der beiden Operanden zu `true` ausgewertet wurde, ansonsten `false`.

Es gibt außerdem den ternären Operator, nämlich `a ? x : y`. Er überprüft den linken Operanden a auf seinen Wahrheitswert und gibt dann den Wert des mittleren Ausdrucks x zurück, wenn der Wahrheitswert `true` ist oder Wert des rechten Ausdrucks y, wenn der Wahrheitswert `false` ist. Der Operator `? :` ist rechtsassoziativ.

5.2.2 Zuweisungsoperatoren

Neben den allgemeinen Operatoren für Ausdrücke, gibt es noch weitere so genannte Zuweisungsoperatoren, bei denen der linke Operand eine Variable und der rechte Operand ein Ausdruck ist, der zum gleichen Datentyp ausgewertet wird wie die Variable. Diese Operatoren sind rechtsassoziativ und existieren in folgender Form:

`=`

Der Operator `=` weist dem linken Operanden den Wert des Ausdrucks auf der rechten Seite zu.

`+=, -=`

Die Operatoren `+=` bzw. `-=` weisen dem linken Operanden den Wert der Variable zuzüglich bzw. abzüglich dem Wert des rechten Operanden zu. Es gelten die Typeinschränkungen der Operatoren `+` und `-`.

`*=, /=, %=`
: Die Operatoren `*=`, `/=` und `\%=` weisen dem linken Operanden den Wert der Variable multipliziert, dividiert oder modulo mit dem Wert des rechten Operanden zu. Es gelten die Typeinschränkungen der Operatoren `*`, `/` und `\%`.

5.2.3 Weitere Operatoren

Neben den vorgestellten gängigen Operatoren, erlaubt DialogOS auch das Benutzen von einigen Abkürzungen und bitweise Operatoren:

`++, -`
: Die Präfix- und Postfixoperatoren `++` und `--` bewirken angewandt auf Variablen oder Konstanten, deren Datentyp eine Zahlendarstellung ist, dass der Wert der Variable um 1 (bwz. 1.0) erhöht oder gesenkt wird. Sie dienen also auch als Zuweisungsoperatoren. Der Unterschied zwischen Postfix- und Präfix-Anwendung ist der, dass Präfixe vor dem Rest des Ausdrucks ausgewertet werden und Postfixe danach. Ist eine Variable a gleich 0, so gilt z.B. `b=3+(a++)` ⇒`b=3`, aber `b=3+(++a)` ⇒`b=4`. In beiden Fällen hat `a` hinterher den Wert 1.

`&`
: Der binäre Operator `&` ist das bitweise Und für Zahlen.

`|`
: Der binäre Operator `|` ist das bitweise Oder für Zahlen.

`^`
: Der binäre Operator `^` ist das bitweise exklusive Oder für Zahlen.

5.2.4 Funktionsaufrufe

Jeder Funktionsaufruf `f(p)` lässt sich auch als Ausdruck verstehen, wobei der Datentyp des Ausdrucks dem Rückgabetyp der Funktion entspricht und die Parameter innerhalb der Funktionsdefinition festgelegt werden. Eine Ausführliche Beschreibung der Funktionssyntax finden Sie in Kapitel 6. Eine Liste der vordefinierten Funktionen finden Sie in Anhang A.

Kapitel 6

Skripte und eigene Funktionen

Aufbauend auf Variablen und Ausdrücken können Sie in DialogOS kleine Programme, sogenannte Skripte schreiben, die zur Laufzeit des Dialogs ausgeführt werden. In diesem Kapitel wird beschrieben, wie die Skriptsprache von Dialog OS aufgebaut ist, und wie Sie Skripte und Funktionen in Ihren Dialogen verwenden.

6.1 Die Skriptsprache

Ein Skript lässt sich als Folge von Befehlen verstehen, die von oben nach unten abgearbeitet wird. Jeder Befehl muss dabei mit einem Semikolon abgeschlossen werden. Typische Anwendungen für Skripte sind komplexe Berechnungen und Modifikationen der Dialogvariablen.

6.1.1 Zuweisungen

Jeder Variable kann ein neuer Wert zugewiesen werden. Dies funktioniert mit den Zuweisungsoperatoren `=`, `+=`, `-=`, `/=` und `*=` (vgl. Kapitel 5.2). Auch die Verwendung der Post- und Präfixoperatoren `++` und `--` bei Variablen lässt sich als Zuweisungsbefehl verstehen, beispielsweise haben die folgenden Befehle alle dasselbe Resultat, nämlich die Erhöhung von `a` um 1:

```
a = a+1;
a += 1;
a++;
++a;
```

6.1.2 Bedingte Ausführung

DialogOS bietet die Möglichkeit der Verwendung von Bedingungen, um Kontrolle darüber zu geben, ob ein Befehl ausgeführt werden soll.

```
if (x == 0)
    prompt = "Sie␣haben␣keinen␣Artikel␣ausgewählt.";
else {
    if (x == 1)
```

```
        prompt = "Sie␣haben␣einen␣Artikel␣ausgewählt.";
    else
        prompt = "Sie␣haben␣" + x + "␣Artikel␣ausgewählt.";
}
```

Das obigen Beispiel weist der Variablen `prompt` in Abhängigkeit von der Variablen `x` einen neuen Wert zu. Beachten Sie dabei die folgenden Details: Der Ausdruck direkt nach dem `if` muss zu einem boolschen Wert (`true` oder `false`) ausgewertet werden können. Der Befehl nach der Bedingung wird nur ausgeführt, wenn die Bedingung wahr war, ansonsten wird der Befehl nach dem `else` ausgeführt. Wenn Sie der Bedingung oder dem `else`-Zweig mehr als einen Befehl unterordnen wollen, müssen Sie diese Liste von Befehlen mit geschweiften Klammern `{ }` einschließen. Der letzte `else`-Zweig besteht aus einer Zuweisung, wobei der neue Wert für `prompt` mit Hilfe eines komplexen Ausdrucks berechnet wird.

6.1.3 Verzweigungen

Wenn Sie eine Variable auf viele verschiedene Werte überprüfen müssen, stellt DialogOS eine übersichtlichere Schreibweise anstelle von vielen, verschachtelten `if`-Anweisungen zur Verfügung:

```
switch (x) {
    case 0:
        prompt = "Sie␣haben␣keinen␣Artikel␣ausgewählt.";
        break;
    case 1:
        prompt = "Sie␣haben␣einen␣Artikel␣ausgewählt.";
        break;
    default:
        prompt = "Sie␣haben␣" + x + "␣Artikel␣ausgewählt.";
        break;
}
```

Am Anfang der `switch`-Anweisung wird die Variable x einmal auf Ihren Wert überprüft. Die Ausführung springt dann direkt an die Stelle der entsprechenden `case`-Marke, oder zur `default`-Marke, falls es für den Wert keine `case`-Marke gibt. Beachten Sie, dass jeder Block mit einer `break`-Anweisung abgeschlossen werden muss, damit die Ausführung nicht einfach weiterläuft sondern zum Ende der `switch`-Anweisung springt.

6.1.4 Lokale Variablen

Skripte bieten darüber hinaus die Möglichkeit, lokale Variablen zu deklarieren, die nur innerhalb des Skripts als kurzfristige Speichermöglichkeit benutzt werden können. Eine Deklaration besteht aus dem Datentyp und dem Namen der neuen Variable. Bei einer Deklaration können Sie, durch ein Gleichheitszeichen abgetrennt, auch direkt den Wert angeben, mit dem die neue Variable initialisiert werden soll. Beispiele für Deklarationen sind

```
struct neueStruktur;
real pi = 3.141;
struct komplexeStruktur = { datum = { wochentag="montag", monat="mai" },
                            wochentage=["montag", "dienstag", "mittwoch"] };
```

Für lokale Variablen stehen Ihnen die gleichen Datentypen wie für Dialogvariablen zur Verfügung (siehe Kapitel 5.1).

6.1.5 Schleifen

DialogOS bietet Ihnen mit Hilfe von Schleifen die Möglichkeit, Befehle wiederholt auszuführen. Es gibt verschiedene Schleifentypen, aber allen ist gemeinsam, dass sie den ihnen untergeordneten Befehl so lange immer wieder ausführen, bis eine bestimmte Bedingung eintritt. Wenn Sie mehrere Befehle in einer Schleife ausführen möchten, müssen Sie diese wiederum in geschweifte Klammern einschließen. Mit einer `break`-Anweisung können Sie allerdings die Ausführung eines Schleife jederzeit abbrechen.

`while`-Schleife

Eine `while`-Schleife ist eine Schleife, die einen Befehlsblock enthält, der so oft ausgeführt wird, bis die Bedingung der Schleife *nicht* mehr erfüllt ist. Dies kann insbesondere auch dann auftreten, wenn die Bedingung vor dem ersten Durchlauf schon nicht erfüllt ist. Die Schleife besteht aus dem Kopf `while(`*Ausdruck*`)` und einem Befehlsblock. Der Ausdruck muss zu einem der beiden Wahrheitswerte `true` oder `false` auswertbar sein.

```
int zahl = 5;
int fak = 1;
while (zahl>1) {
    fak = fak * zahl;
    zahl--;
}
```

`do-while`-Schleife

Eine `do-while`-Schleife ist eine Schleife, die analog zur while-Schleife einen Befehlsblock enthält, der so oft ausgeführt wird, bis die Bedingung nicht mehr erfüllt ist. Allerdings wird die Schleife mindestens einmal durchlaufen, da die Bedingung erst am *Ende* der Schleife überprüft wird.

```
fak = 1;
do {
    fak = fak * zahl;
    zahl--;
} while(zahl>1);
```

`for`-Schleife

Eine `for`-Schleife ist eine Schleife, die typischerweise die Zahl der Durchläufe in einer lokalen Variablen mitzählt. Der Kopf besteht aus drei Teilen: einer Initialisierung der lokalen Variablen, einer Bedingung und einer Anweisung, die *nach* jeder Abarbeitung des Befehlblocks ausgeführt wird. In der Initialisierung können Sie sogar eine neue lokale Variable definieren, die nur innerhalb dieser Schleife gültig ist.

```
int fak = 1;
for (int zahl = 5; zahl>1; zahl--)
    fak = fak * zahl;
```

6.2 Funktionen

Wenn Sie bestimmte Befehlsfolgen häufiger an verschiedenen Stellen ausführen wollen, ist es lästig und fehleranfällig, das entsprechende Skript an die verschiedenen Stellen zu kopieren. Stattdessen bietet Ihnen DialogOS die Möglichkeit, eigene Funktionen zu definieren, die Sie immer wieder aufrufen können. Funktionen können Parameter haben, die bei jedem Aufruf andere Werte annehmen können. Außerdem können Funktionen einen Wert als Ergebnis zurückgeben.

Eine Funktion besteht aus einem Funktionskopf, der Rückgabetyp, Funktionsname und Parameter in der genannten Reihenfolge enthält, und einem Funktionsrumpf, der wie ein Skript aufgebaut ist und *immer* mit geschweiften Klammern vor dem ersten und nach dem letzten Befehl gekennzeichnet wird, auch wenn es sich um einen einzelnen Befehl handelt. Dabei werden die Parameter als Ausdruck der Form (Typ Name, ..., Typ Name) hinter den Funktionsnamen geschrieben. Ist der Rückgabetyp nicht `void` ("nichts"), so muss die Funktion einen Wert zurückliefern.

Die Syntax für eine Funktion sieht also wie folgt aus:

Typ Funktionsname `(` *Parameter* `) {`
    Funktionsrumpf
`}`

konkret z.B.

```
int Summe(int a, int b) {
    return a+b;
}
```

wobei die Parameter `a`, `b` und der Rückgabewert vom Datentyp `int` sind.

Innerhalb von Funktionen ist es wichtig, dass am Ende ein Wert zurückgegeben wird, der dem festgelegten Rückgabetyp entspricht. Dies gilt für alle Datentypen außer `void`. Eine Rückgabeanweisung besteht immer aus dem Befehl `return` und einem anschließenden Ausdruck, der zu einem Wert des Datentyps ausgewertet wird. Wenn Sie eine `void`-Funktion vorzeitig verlasen wollen, können Sie dafür den `return`-Befehl ohne Argument verwenden. Eine Rückgabe sieht also beispielsweise wie folgt aus:

```
return 3.141;
return a+b;
return;
```

6.3 Skriptknoten

Sie können ein neues Skript in Ihren Dialog einfügen, indem Sie einen *Skript*-Knoten in den Arbeitsbereich ziehen. Durch Doppelklick auf den *Skript*-Knoten erhalten Sie ein Fenster, in dem Sie neben den allgemeinen Knoteneigenschaften auch das Skript bearbeiten können, das mit diesem Knoten verbunden sein soll. Bei der Ausführung des Dialogs wird dann beim Erreichen des *Skript*-Knotens das angehängte Skript ausgeführt und der Dialog danach über den Ausgang des *Skript*-Knotens fortgesetzt.

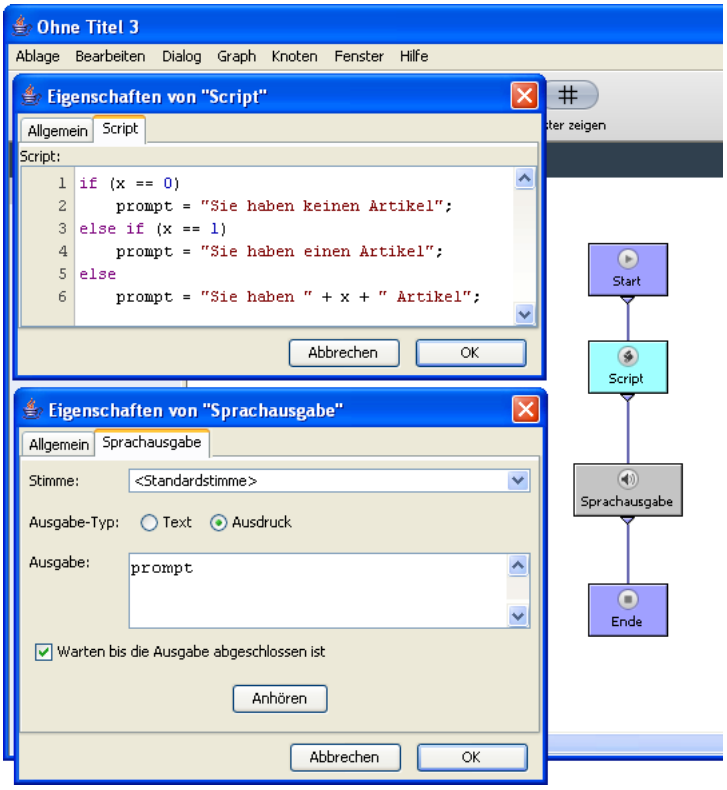

Abbildung 6.1: Dialog mit *Skript*-Knoten

So können Sie z.B. mit Hilfe des Skripts die nächste Systemausgabe berechnen und in einer Variablen speichern, und dann in der Sprachausgabe auf diese Variable zugreifen. Beachten Sie, dass Sie für den Zugriff auf die Variable den Ausgabetyp am *Sprachausgabe*-Knoten auf *Ausdruck* stellen müssen.

6.4 Eigene Funktionen

Eigene Funktionen können Sie an zwei Stellen in DialogOS definieren: entweder innerhalb eines *Skript*-Knotens oder für den ganzen Dialog.

Wenn Sie eine Funktion nur innerhalb eines einzigen Skripts benötigen, können Sie diese einfach am Anfang des Skripts definieren.

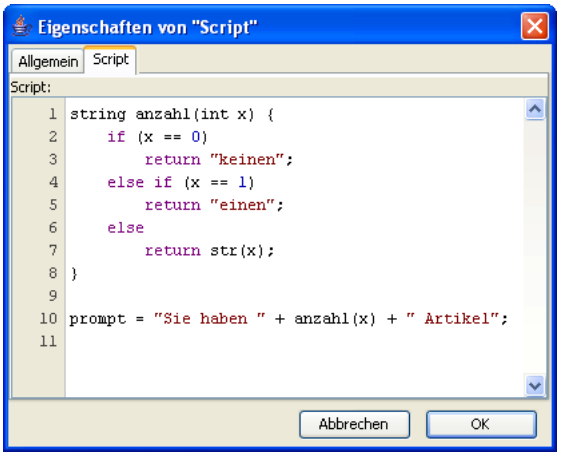

Abbildung 6.2: Skriptknoten mit Funktionsdefinition im Skript

Funktionen, die Sie an verschiedenen Stellen im Dialog verwenden wollen, können Sie an zentraler Stelle definieren, indem Sie im Menü `Graph` den Befehl `Funktionen` wählen.

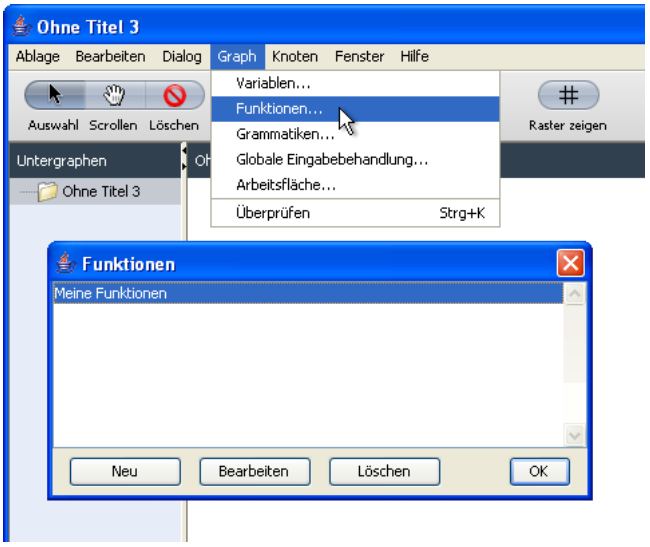

Abbildung 6.3: Dialogweite eigene Funktionen

Im Fenster *Funktionen* können Sie Module anlegen und löschen. Jedes Modul kann beliebig viele Funktionsdefinitionen enthalten. Die Funktionen, die in einem solchen Modul definiert sind, sind an jeder Stelle im Dialog verfügbar, d.h. innerhalb von *Skript*-Knoten aber auch innerhalb von Ausdrücken in allen anderen Knotentypen.

Kapitel 7

Muster

Muster haben in DialogOS zweierlei Funktion: Sie dienen auf der einen Seite als Kontrollstruktur, da durch sie entschieden werden kann, wie ein Dialog verläuft, auf der anderen Seite können sie einzelne Informationen einer Eingabe in Variablen speichern, um so im späteren Dialogverlauf weiterhin Zugriff auf die Information zu haben.

Es gibt verschiedene Varianten von Mustern, die man in Eingabeknoten verwendet. Generell wird ein Dialog an dem jeweils entsprechenden Ausgang fortgesetzt, sobald der Wert einer Eingabe auf ein Muster passt. Die Muster werden dabei von oben nach unten überprüft. Die folgende Liste soll einen Überblick über Mustervarianten und deren Vor- und Nachteile geben:

Konstante Werte
: Es ist möglich, konstante Zeichenketten, Zahlen- und Wahrheitswerte als Muster zu verwenden. Diese passen genau dann auf eine Eingabe, wenn die Eingabe den gleichen Wert hat. Es wird dann keine Information gespeichert und der Dialog am entsprechenden Ausgang fortgesetzt. Beispiele: `1`, `3.141`, `"Autobahn"`, `true`.

Variablen
: Benutzt man Variablen als Muster, so wird der Wert der Eingabe in die Variable geschrieben und der Dialog wird am entsprechenden Ausgang fortgesetzt. Man sollte allerdings der verwendete Datentyp der Variable beachten.

Universeller Platzhalter
: Ein einfacher Unterstrich (`_`) als Muster wird in der Regel am Ende der Liste von Eingabemustern verwendet, da es zu jeder Eingabe passt. Es speichert keinerlei Informationen, sondern setzt den Dialogverlauf entlang des entsprechenden Ausgangs fort.

Listen
: Listen funktionieren als Muster genauso wie Werte. Es gibt allerdings zusätzlich die Möglichkeit, ein Listenmuster mit dem Operator `::` darzustellen, wenn nur die Anzahl der Elemente oder der Wert bestimmter Elemente von Bedeutung ist. Beispielsweise passt `_::_::[]` auf alle Listen mit genau zwei Elementen, während `_::_::_` auf alle Listen mit mindestens zwei Elementen passt.

Strukturen
: Strukturen funktionieren ebenfalls genauso wie Werte. Allerdings passt eine Struktur auf alle Eingabe-Strukturen, die mindestens die gleiche Menge von Informationen beinhalten. Es ist daher sinnvoll, Strukturen mit mehr Informationen weiter oben in die Liste von Mustern zu schreiben,

da Eingaben sonst vorher bereits auf eine Struktur mit weniger Informationen passen könnten. Beispielsweise passt das Muster `{ help=_ }` sowohl auf die Struktur `{ help="control" }` als auch auf die Struktur `{ help="control", route="Autobahn" }`

Variablen mit Werten

Es ist möglich, sowohl eine Variable als auch einen Wert (bzw. eine Liste oder Struktur) als Muster anzugeben. Der Wert der Eingabe wird nun, wenn er auf das Muster passt, in die angegebene Variable geschrieben und dann erst der Dialog am entsprechenden Ausgang fortgeführt. Ein solches Variablenmuster hat die Form *Variable* `as` *Wert*. Beispielsweise speichert `chosen as { help=_ }` eine eingegebene Struktur, die ein Element mit dem Namen *help* hat in die Variable *chosen* und setzt anschließend den Dialog am entsprechenden Ausgang fort.

Reguläre Ausdrücke

Ist die Eingabe eine Zeichenkette, so ist es möglich, einen regulären Ausdruck auf die Eingabe anzuwenden und die gefundenen Daten anschließend in Variablen zu speichern. Ein solches Muster hat die Form */regexp/* `=` (*Variablen*). Beispielsweise prüft `/ja.*/`, ob eine Zeichenkette mit den Buchstaben "ja" beginnt. Sie können Teile des regulären Ausdrucks in runde Klammern einschließen und den Inhalt der Klammern in einer Variablen speichern. Der reguläre Ausdruck `/Ich möchte ein Lied von (.*)/=(artist)` speichert z.B. den Namen eines Künstlers in die Variable *artist*. Sie können mit einem Ausdruck auch mehrere Variablen füllen. Die einzige Voraussetzung ist, dass der reguläre Ausdruck mindestens so viele Klammerausdrücke enthält, wie Variablen angegeben sind. Beispielsweise speichert `/Ich möchte (.*) von (.*)/=(song,artist)` den Inhalt des ersten Platzhalters in die Variable *song* und den des zweiten Platzhalters in die zweite Variable, also *artist*.

Kapitel 8

Erweiterte Sprachsteuerung

Nachdem Sie sich mit den Konzepten für Ausdrücke, Muster und Grammatiken vertraut gemacht haben, wird im Folgenden beschrieben, wie Sie damit komplexe Sprachdialoge gestalten können. Dazu sollen die erweiterten Einstellungsmöglichkeiten der *Spracheingabe*- und *Sprachausgabe*-Knoten, die in Kapitel 3 beschrieben wurden, näher beleuchtet werden.

8.1 Dynamische Sprachausgabe

Sprachausgabeknoten bieten zwei Formate für den auszugebenden Text: *Text* und *Ausdruck*. Wenn Sie als Format *Ausdruck* wählen, können Sie einen komplexen Ausdruck angeben, aus dem zur Laufzeit der auszugebende Text berechnet wird. Dies bietet Ihnen die Möglichkeit, den Text dynamisch an die Dialogsituation anzupassen, z.B. indem Sie Bezug auf Dialogvariablen nehmen. Dadurch erreichen Sie eine weitaus größere Flexibilität bei der Beschreibung der Systemausgaben.

8.2 Komplexe Spracheingabe

8.2.1 Grammatiken

An jedem Spracheingabeknoten des Dialogmodells, muss vordefiniert werden, welche Äußerungen möglich sind. Eine Menge formaler Regeln, die diese Äußerungen beschreiben, nennt man auch Grammatik.

Grammatiken haben zwei Vorteile gegenüber einfachen Spracherkennungsmustern, wie sie in Kapitel 3 vorgestellt wurden: Sie können wesentlich komplexere Beschreibung der Äußerung enthalten und sie sind universell verfügbar, d.h. die gleiche Grammatik kann an mehreren Spracherkennungsknoten verwendet werden und muss nicht immer wieder neu definiert werden.

Das Verwalten der Grammatiken im Dialogmodell ist über den Menüpunkt `Grammatiken` im Menü `Graph` möglich.

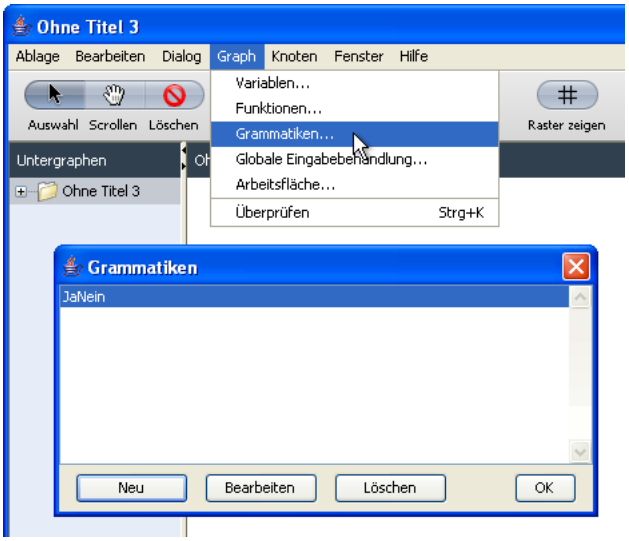

Abbildung 8.1: Definition von Grammatiken

Eine Grammatik besteht aus einem Kopf, in dem Meta-Informationen zur Grammatik gespeichert werden, und einem Rumpf, in dem die Regeln stehen, mit denen eine Benutzereingabe abgeglichen wird. Eine Beispielgrammatik könnte wie folgt aussehen:

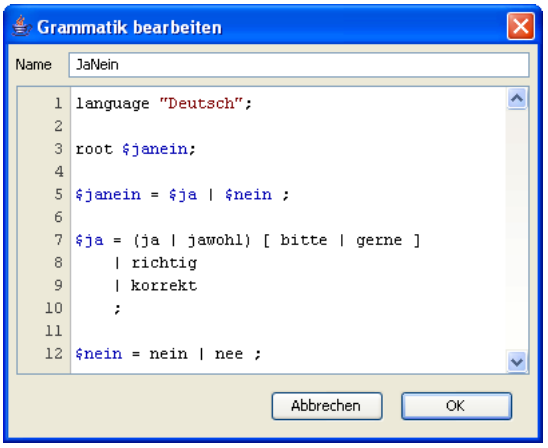

Abbildung 8.2: Bearbeitung einer Grammatik

In jeder Grammatik müssen Sie genau eine Grundregel (*root*, hier: `$janein`) festlegen, über die bestimmt wird, welche Aussagen ein Benutzer machen kann. Regelnamen erkennen Sie an dem führenden `$`-Zeichen. Die Definition einer Regel besteht aus dem Regelnamen, gefolgt von einem `=`-Zeichen und der *Expansion* der Regel, d.h. der Liste von Äußerungen, die durch die Regel beschrieben werden. Diese Expansion kann wiederum Verweise auf andere Regeln enthalten, die ebenfalls expandiert werden. So verweist die Regel `$janein` in obigem Beispiel auf die Regeln `$ja` und `$nein`.

Mit dem Zeichen `|` werden alternative Aussageteile voneinander getrennt, d.h. der Benutzer muss entweder die linke *oder* die rechte Äußerung sagen.

Eine Folge von Worten (auch *Sequenz* genannt), muss der Benutzer genau in der Reihenfolge sagen - nicht ein Wort weniger und nicht ein Wort mehr.

Wenn Sie innerhalb einer Sequenz eine Alternative beschreiben wollen, müssen Sie diese in runde Klammern einschließen, um die Gruppierung deutlich zu machen. Es gilt nämlich: `a b | c d` ist dasselbe wie `(a b) | (c d)`, aber etwas ganz anderes als `a (b | c) d`.

Die eckigen Klammern `[` und `]` markieren Anfang und Ende von optionalen Aussageteilen, d.h. der Benutzer kann dieses Wort (oder die Sequenz) weglassen. Die verschiedenen Klammerungen können ineinander verschachtelt werden, um beliebig komplexe Ausdrucksbeschreibungen zu erzeugen.

Leerzeichen, Tabulatoren und Zeilenumbrüche dienen lediglich zur Trennung von Worten und haben ansonsten keine Bedeutung. Satzzeichen oder Sonderzeichen sind nicht erlaubt.

Die Zeichen `+` und `*` haben eine besondere Bedeutung. Sie signalisieren, dass ein Wort oder eine Sequenz beliebig oft wiederholt werden kann, wobei das Wort `+` mindestens einmal, bei `*` ggf. gar nicht vorkommen muss.

Die Benutzung eines Regelnamens auf der rechten Seite bewirkt, dass an dieser Stelle die Definition dieser Regel eingesetzt wird. Die obige Grammatik beschreibt also in der Regel `$janein`, dass der Benutzer entweder den Inhalt der Regel `$ja` oder den Inhalt der Regel `$nein` sagen kann. `$ja` wiederum beschreibt die Worte "ja" oder "jawohl", gefolgt von dem Wort "bitte" oder "gerne", wobei dieser zweite Teil auch ganz wegfallen kann. Alternativ kann der Benutzer auch "richtig" oder "korrekt" sagen. Insgesamt beschreibt die Grammatik also die Äußerungen "ja", "ja bitte", "ja gerne", "jawohl", "jawohl bitte", "jawohl gerne", "richtig", "korrekt", "nein" und "nee".

Sie können im Kopf der Grammatik die Sprache angeben, die für die Erkennung verwendet werden soll (hier: `language "Deutsch"`). Wenn die Definition der Sprache in der Grammatik fehlt, muss die Sprache im jeweiligen Spracherkennungsknoten ausgewählt werden.

8.2.2 Spracherkennung mit Grammatiken

Die Spracherkennung mit Grammatiken läuft in einem zweistufigen Verfahren ab. Zunächst wird mit Hilfe der Grammatik die Spracherkennung konfiguriert und dann gestartet. Das Ergebnis der Spracherkennung ist eine Zeichenkette (`string`), die eine durch die Grammatik beschriebene Äußerung enthält. Diese Zeichenkette wird dann mit den von Ihnen angegebenen Mustern verglichen.[1]

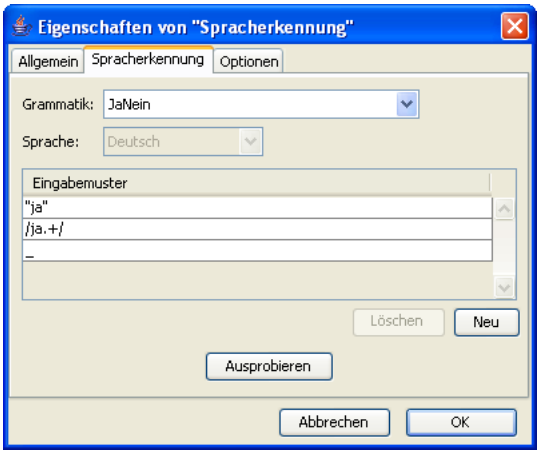

Abbildung 8.3: Muster für die Spracherkennung mit Grammatik

[1] Achtung: Wenn Sie die Grammatik *automatisch generieren* lassen, müssen Sie als Muster einfach die zu erkennenden Worte im Klartext hinschreiben. Wenn Sie vordefinierte Grammatiken verwenden, müssen Sie jedoch die in Kapitel 7 beschriebene Syntax für Muster verwenden.

Die Muster werden von oben nach unten der Reihe nach getestet. Das erste Muster, das passt, wird ausgewählt und der Dialog über den zu diesem Muster gehörenden Ausgang des *Spracheingabe*-Knotens fortgesetzt. Im obigen Beispiel passt das erste Muster, wenn der Benutzer genau "ja" gesagt hat (konstantes Muster), das zweite, wenn er "ja" gefolgt von weiteren Worten gesagt hat (regulärer Ausdruck) und das dritte auf alle anderen Äußerungen, die die Grammatik beschreibt.

8.2.3 Tags

Wenn Sie große Grammatiken schreiben, werden Sie schnell feststellen, dass es sehr lästig ist, die möglichen Äußerungen an jedem *Spracheingabe*-Knoten noch einmal durch Muster beschreiben zu müssen. In der Regel erfordert dies nämlich den Gebrauch komplexer, regulärer Ausdrücke, die einen Großteil des in der Grammatik kodierten Wortschatzes wiederholen.

Deshalb unterstützt DialogOS semantische Tags innerhalb von Grammatiken. Tags sind Anweisungen, die im Zuge des Spracherkennungsprozess einen Wert berechnen und diesen anstelle der erkannten Wortfolge als Ergebnis der Spracherkennung liefern. Tags werden durch geschweifte Klammern eingeschlossen, ein einzelnes `$`-Zeichen bezieht sich innerhalb eines Tags dabei auf den aktuellen Wert der Regel, in dem der Tag steht.:

```
language "Deutsch";
root $janein;

$janein = $ja   { $ = "ja" }
        | $nein { $ = "nein" }
        ;

$ja = ...
$nein = ...
```

Im obigen Beispiel bewirken die Tags, dass das Ergebnis der Spracherkennung genau "ja" ist, wenn die Spracherkennung irgendeine Expansion der Regel `$ja` erkannt hat. Dabei ist völlig egal, wie genau die Expansion der Regel aussieht und welche Worte tatsächlich erkannt wurden. Am Spracheingabeknoten müssen so nur noch die konstanten Muster `"ja"` und `"nein"` überprüft werden.

Aber Tags können noch mehr. Sie können beliebig komplexe Ausdrücke berechnen und sich auf das Ergebnis von Tags in untergeordneten Regeln beziehen.

```
language "Deutsch";
root $start;

$start = $anzahl { $ = $anzahl }
       | nein    { $ = 0 } ;

$anzahl =  { $ = 0 } ( ja  { $ = $+1 } ) + ;
```

In diesem Beispiel kann der Benutzer entweder "nein" oder beliebig oft (aber mindestens einmal) "ja" sagen. Die Tags sorgen dafür, dass als Ergebnis der Erkennung die Anzahl der Ja-Worte geliefert wird. Betrachten Sie zunächst die Regel `$anzahl`. Durch einen Tag wird der Ergebniswert der Regel auf `0` gesetzt. Danach kann das Wort "ja" erkannt werden, wobei der folgende Tag dafür sorgt, dass der Ergebniswert der Regel `$anzahl` um eins erhöht wird. Das `+` hinter der Sequenz aus Wort und Tag sorgt dafür,

dass diese Sequenz beliebig oft wiederholt werden kann. Nun zur Regel `$start`. Sie kann zum einen aus der Expansion der Regel `$anzahl` bestehen. In diesem Fall übernimmt der nachfolgende Tag den Wert der Regel `$anzahl` als Ergebniswert für die Regel `$start`. Im anderen Fall (wenn der Benutzer "nein" sagt), wird der Wert explizit auf `0` gesetzt.

Das Ergebnis der Spracherkennung muss also nicht zwangsläufig eine Zeichenkette sein. Wenn Sie Tags verwenden, um den Ergebniswert zu berechnen, können Sie alle Datentypen, Operatoren und Funktionen der DialogOS Skriptsprache verwenden.

Sie können Tags aber auch einfach benutzen, um den relevanten Teil einer komplexen Äußerung auszufiltern und an die Dialogsteuerung zurückgeben:

```
language "Deutsch";
root $start;

$start = ich möchte nach $ort { $ = $ort } ;
$ort =  Hamburg | Stuttgart | München | Saarbrücken ;
```

Durch ein Variablen-Muster am Spracheingabeknoten können Sie nun z.B. den genannten Ortsnamen ganz einfach in einer Variablen speichern.

Kapitel 9

Weitere Knotentypen

Sie haben mittlerweile verschiedene Knotentypen für die Spracheingabe, die Sprachausgabe, die Steuerung eines Lego Mindstorms Roboters sowie die Ausführung von Skripten kennengelernt. DialogOS bietet darüber hinaus eine ganze Reihe von weiteren, speziellen Knotentypen, mit denen Sie den Dialogablauf beeinflussen können. Diese Knoten machen allesamt regen Gebrauch von Variablen, Ausdrücken und Mustern. Sie sollten mit diesen Konzepten vertraut sein, bevor Sie dieses Kapitel lesen.

Jeder dieser Knotentypen hat spezielle Parameter, die Sie wie gewohnt bearbeiten können, indem Sie einen Doppelklick auf das Knotensymbol in der Arbeitsfläche machen.

9.1 *Bedingungs*-Knoten

An einem Bedingungsknoten verzweigt sich der Ablauf des Dialogs in Abhängigkeit von einer frei definierbaren Bedingung. Die Bedingung selbst kann ein beliebiger Ausdruck sein, der zur Laufzeit ausgewertet wird. Das Ergebnis der Auswertung muss ein boolscher Wert sein.

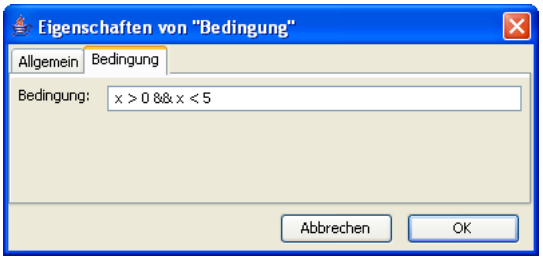

Abbildung 9.1: Einstellungen für Bedingungsknoten

Bedingungsknoten haben immer genau zwei Ausgänge: “wahr” und “falsch”. Je nachdem welches Ergebnis die Auswertung der Bedingung liefert, wird der Dialogablauf über den entsprechenden Ausgang fortgesetzt.

9.2 *Warten*-Knoten

Warteknoten erlauben, im Dialogablauf eine beliebige Verzögerung einzubauen. Die Länge der Ausführungspause wird durch den Parameter des Knotens angegeben. Dies kann ein beliebiger Ausdruck sein,

dessen Auswertung als Ergebnis eine Ganzzahl (codeint) liefern muss. Diese Zahl gibt die Pausenlänge in Millisekunden an.

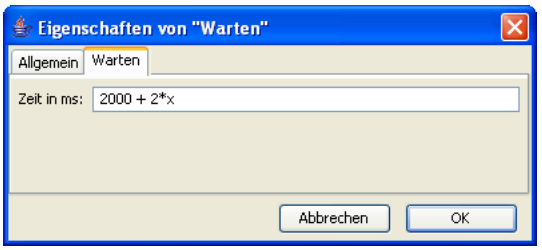

Abbildung 9.2: Einstellungen für Warteknoten

Warteknoten haben immer genau einen Ausgang.

9.3 Variablen

DialogOS bietet zwei Arten von Variablenknoten: einen zum Setzen einer Variablen und einen zum Testen eines Variablenwerts.

9.3.1 *Variable setzen*

Die speziellen Parameter dieses Knotentyps sind zum einen die Variable, der ein neuer Wert zugewiesen werden soll, sowie ein Ausdruck, der den neuen Wert bestimmt. Zur Ausführungszeit des Dialogs wird dieser Ausdruck ausgewertet und das Ergebnis in die ausgewählte Variable geschrieben. Ist keine Variable ausgewählt oder wird kein Ausdruck angegeben, so tritt an diesem Knoten ein Laufzeitfehler auf und die Ausführung wird abgebrochen.

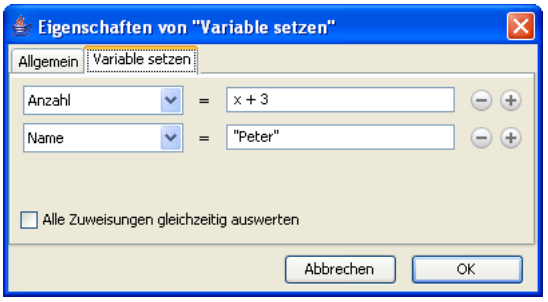

Abbildung 9.3: Einstellungen für *Variable setzen*-Knoten

9.3.2 *Variable testen*

Sie können eine ausgewählte Variable auf verschiedene Werte testen. Diese Werte werden durch eine Liste von Mustern beschrieben, wobei der Knoten für jedes Muster genau eine Ausgangskante erhält. Zur Laufzeit wird der aktuelle Wert der ausgewählten Variablen von oben nach unten mit der Liste der angegebenen Muster verglichen. Das erste passende Muster wird ausgewählt und die Ausführung über die dazugehörige Ausgangskante fortgesetzt.

Optional können Sie eine zusätzliche Kante einfügen, über die der Dialog fortgesetzt wird, wenn keines der angegebenen Muster passt. Wenn Sie auf diese Option verzichten und zur Laufzeit keines der Muster auf den aktuellen Wert der Variablen passt, so tritt ein Laufzeitfehler auf und die Ausführung wird abgebrochen.

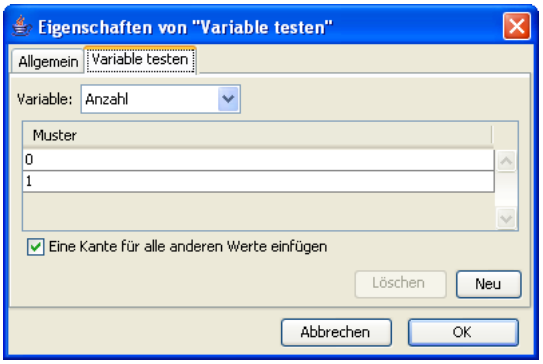

Abbildung 9.4: Einstellungen für *Variable testen*-Knoten

9.4 Sprünge

In komplexen Dialogsystemen kann es vorkommen, dass Sie von verschiedenen Stellen im Dialog an eine bestimmte Stelle (z.B. zum Anfang des Dialogs) zurückkehren wollen. Viele rücklaufende Kanten müssten dafür zurück zum Startknoten gezogen werden und würden dabei den Dialog kreuzen und somit die Übersichtlichkeit reduzieren. Aus diesem Grund unterstützt DialogOS explizite Sprünge. Sprünge können Sie sich also als eine Art unsichtbare Kante vorstellen, die zwischen zwei Knoten verläuft.

9.4.1 *Sprungziel*

Sprungziel-Knoten können an jeder Stelle im Dialog eingefügt werden. Sie dienen als Ziel von Sprungknoten, können aber auch ganz normal durchlaufen werden, d.h. Sie können auch direkte Kanten von anderen Knoten zu Sprungziel-Knoten ziehen. Der Sprungzielknoten führt keine eigene Aktion aus, er ist lediglich als organisatorische Einheit zu verstehen. Ein Sprungzielknoten hat genau einen Ausgang, über den der Ablauf des Dialogs fortgesetzt wird, egal ob der Knoten durch eine direkte Kante oder durch einen Sprung erreicht wurde.

9.4.2 *Sprung*

Sprung-Knoten sind der Ausgangspunkt eines Sprungs. Als Parameter des Knotens können Sie das Sprungziel auswählen, an dem der Dialogablauf fortgeführt werden soll. Demzufolge haben *Sprung*-Knoten keinen direkten Ausgang. Stattdessen wird die Dialogkontrolle bei der Ausführung direkt an das Sprungziel übergeben und der Dialog über dessen Ausgang fortgeführt.

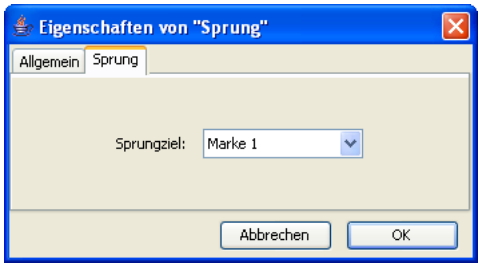

Abbildung 9.5: Einstellungen für *Sprung*-Knoten

Kapitel 10

Große Dialogsysteme organisieren

Wenn Sie größere Dialogsysteme bauen wollen, die aus vielen Knoten bestehen, ist es wichtig, dass Sie das Dialogmodell so aufbauen, dass es übersichtlich und strukturiert bleibt. DialogOS unterstützt Sie bei der Organisation des Dialogmodells durch viele nützliche Funktionen.

10.1 Vergrößern der Arbeitsfläche

Zunächst werden Sie feststellen, dass die standardmäßig vorgesehene Größe des Arbeitsbereichs nicht mehr ausreicht, um alle Knoten übersichtlich darin platzieren zu können. Sie können den Arbeitsbereich jedoch jederzeit vergrößern, indem Sie im Menü `Graph` den Befehl `Arbeitsfläche` auswählen.

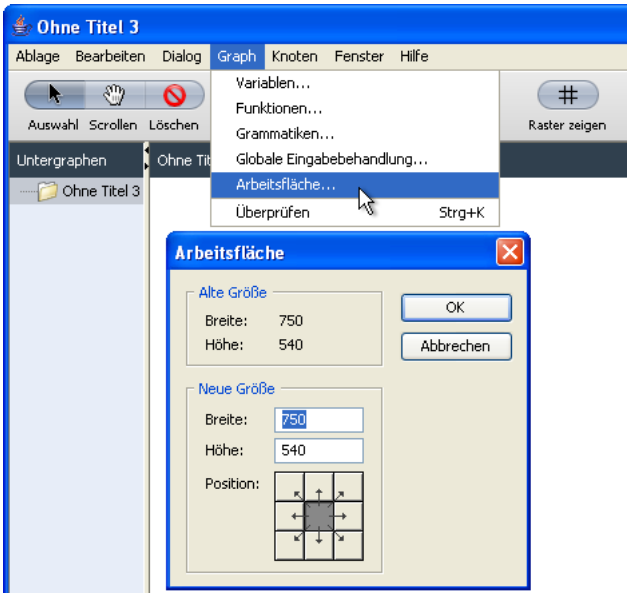

Abbildung 10.1: Vergrößern der Arbeitsfläche

In dem Einstellungsfenster können Sie die neue Größe der Arbeitsfläche auswählen, sowie die Richtung, in die die Arbeitsfläche erweitert wird.

10.2 Kommentare einfügen

Sie können direkt in der Arbeitsfläche Kommentare einfügen, um zu beschreiben, was an dieser Stelle im Dialog passiert. Auf diese Weise können Sie sehr einfach die Logik des Dialogablaufs dokumentieren. Um einen Kommentar einzufügen, klicken Sie mit der rechten Maustaste in die Arbeitsfläche und wählen Sie aus dem sich öffnenden Popup-Menü den Befehl `Kommentar`. An der Stelle, wo sich der Mauszeiger befindet, wird nun eine Kommentarfeld eingefügt. In dieses können Sie nach einem Klick mit der Maus beliebigen Text eintragen.

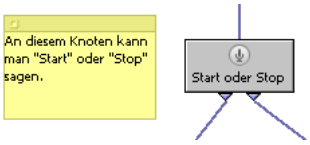

Abbildung 10.2: Kommentare

Sie können das Kommentarfeld verschieben, indem Sie mit der Maus in den Rahmen am oberen Rand des Kommentarfelds klicken und die Maus dann an die gewünschte Stelle bewegen. Ein Klick in das rechteckige Feld in der linken oberen Ecke des Kommentars löscht diesen aus der Arbeitsfläche.

10.3 Knoten ausrichten

Für die optische Übersichtlichkeit eines Modells kann es sehr hilfreich sein, zusammengehörige Knoten an einer Achse auszurichten, so dass diese optisch eine Einheit bilden. Um mehrere Knoten auszuwählen, klicken Sie diese bei gedrückter Shift-Taste an oder ziehen Sie mit der Maus ein Auswahlrechteck um die Knoten, die Sie auswählen möchten. Wählen Sie dann im Menü `Knoten` den Befehl `Ausrichtung`. Nun können Sie die Knoten entlang der horizontalen oder vertikalen Achse gemäß der Mitte oder einer Außenkante bündig ausrichten.

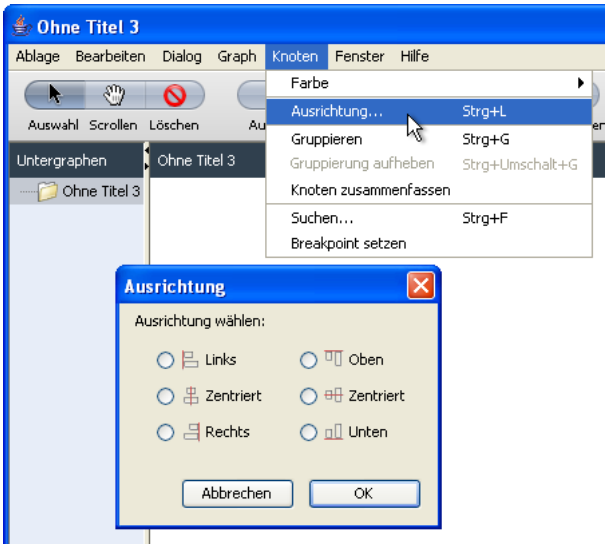

Abbildung 10.3: Ausrichtung mehrerer Knoten entlang einer Achse

10.4 Knoten gruppieren

Sie können mehrere Knoten und Kommentare gruppieren, so dass sich diese Elemente in der Arbeitsfläche wie eine Einheit verhalten. Wählen Sie dazu im Menü `Knoten` den Befehl `Gruppieren`. Ab sofort wird automatisch immer die ganze Gruppe ausgewählt, wenn Sie ein Element der Gruppe auswählen. Insbesondere können Sie die Gruppe nur noch als Ganzes bewegen, wobei die Ausrichtung der Objekte untereinander beibehalten wird. Sie können allerdings nach wie vor die Einstellungen einzelner Knoten durch Doppelklick auf den Knoten bearbeiten.

Gruppen können selbst wieder Gruppen enthalten, d.h. Sie können bei vier Knoten erst Knoten 1 und 2 sowie Knoten 3 und 4 gruppieren und dann eine Gruppe aus diesen beiden Gruppen bilden.

Um eine Gruppierung wieder aufzulösen, wählen Sie im Menü `Knoten` den Befehl `Gruppierung aufheben`.

10.5 Untergraphen

Wenn die Zahl der Knoten zu groß wird, um diese übersichtlich darzustellen, stellt Ihnen DialogOS ein weiteres Hilfsmittel zur Verfügung. Sie können eine Menge von zusammengehörenden Knoten in einem eigenen Unterdialog kapseln. So können Sie große Dialogsysteme aufbauen, die aus mehreren kleinen Teildialogen bestehen.

Um einen neuen Unterdialog anzulegen, ziehen Sie einen Knoten vom Typ *Graph* aus der Knotenleiste am rechten Fensterrand in die Arbeitsfläche. Beobachten Sie dabei die Untergraph-Liste am linken Fensterrand. Sobald Sie den neuen *Graph*-Knoten hinzugefügt haben, erscheint Ihr neuer Knoten als Eintrag in der Liste der Untergraphen. Da Untergrapen selbst wieder Untergraphen enthalten können, können Sie beliebig tief geschachtelte Dialogstrukturen aufbauen. Die Liste der Untergraphen ist deshalb als hierarchische Struktur dargestellt, in der Sie einzelne Ebenen durch Klick mir der Maus auf- und zuklappen können.

Wenn Sie auf einen Eintrag in der Untergraphen-Liste klicken, wird der Inhalt dieses (Teil-)Graphen in der Arbeitsfläche dargestellt. So können Sie schnell zwischen den verschiedenen Untergraphen und Ihrem Hauptgraphen hin und her wechseln. Alternativ können Sie einen Doppelklick auf den Graph-Knoten machen, um den dazugehörigen Unterdialog in einem eigenen Fenster zu öffnen.

Untergraphen sind aus Sicht des Hauptdialogs einfach spezielle Knoten, in denen irgendeine komplexe Funktion ausgeführt wird. Sie verbinden einen Teildialog wie jeden anderen Knoten, indem Sie im Hauptdialog eine Kante von einem Knoten zum Graph-Knoten ziehen. Bei der Ausführung wechselt die Kontrolle mit dem Erreichen des Graph-Knotens zum Start-Knoten des Unterdialogs. Aber wie kehrt die Kontrolle vom Teildialog zum Hauptdialog zurück?

10.5.1 Return-Knoten

Das Ende eines Unterdialogs wird durch Knoten vom Typ *Return* markiert. Return-Knoten haben wie die Ende-Knoten des Hauptdialogs keine eigenen Ausgänge. Stattdessen geben Sie die Kontrolle an den Hauptdialog zurück.

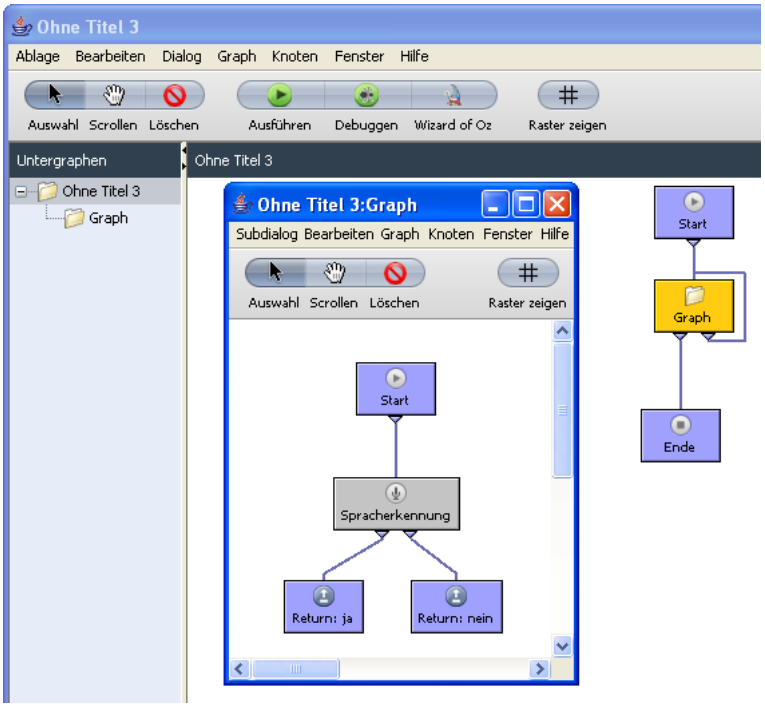

Abbildung 10.4: Ein Untergraph mit zwei Ausgängen

Untergraphen können beliebig viele Return-Knoten haben. Für jeden Return-Knoten des Untergraphs erhält der Graph-Knoten im Hauptdialog einen Ausgang. Auf diese Weise kann der Hauptdialog unterscheiden, in welchem Zustand der Unterdialog beendet wurde. Wenn der Unterdialog im ersten Return-Knoten endete, wird der Hauptdialog über die erste Kante des Graph-Knoten fortgesetzt. Wurde der Unterdialog im zweiten Return-Knoten abgeschlossen, fährt der Hauptdialog über den zweiten Ausgang des Graph-Knoten fort.

Abbildung 10.4 verdeutlicht die Zusammenhänge. Das Model *Ohne Titel 3* enthält einen Unterdialog namens *Graph*. Sie sehen diesen in der Untergraphliste am linken Fensterrand als untergeordnetes Element zum Hauptdialog. Im Vordergrund sehen Sie den Inhalt des Untergraphen in einem eigenen Fenster, während die Arbeitsfläche im Hintergrund den Inhalt des Hauptdialogs zeigt. Wie Sie sehen hat der Untergraph zwei Return-Knoten. Dementsprechend hat der Graph-Knoten im Hauptdialog auch zwei Ausgänge. Der zweite Ausgang ist dabei wieder mit dem Graph-Knoten selbst verbunden, d.h. hier läuft der Dialog in eine Schleife. Der Untergraph besteht in dem Beispiel lediglich aus einem Spracheingabe-Knoten, der die Kommandos *ja* und *nein* erkennt. Damit ist die Funktion des Gesamtdialogs klar. Solange Sie *nein* sagen, läuft der Dialog durch die Schleife immer wieder in den Untergraphen und startet erneut die Spracherkennung. Erst wenn Sie einmal *ja* sagen, wird der Graph-Knoten über die erste Kante verlassen und endet in dem damit verbundenen Ende-Knoten.

Untergraphen können ihre eigene Menge von Variablen, Funktionen und Grammatiken enthalten und bilden damit vollständige Teildialoge.

10.6 Prozeduren

Prozedur-Knoten beschreiben - ähnlich wie *Graph*-Knoten - Teildialoge. Im Gegensatz zu Graph-Knoten können Prozedur-Knoten jedoch nicht direkt durch Kanten verbunden werden. Prozedurknoten reprä-

sentieren vielmehr Teildialoge, die von mehreren verschiedenen Stellen aus angesprungen werden können, wobei die Ausführung des Dialogs nach Beendigung des Teildialogs wieder an die ursprüngliche Aufrufstelle zurückkehrt. Aus diesem Grund haben Prozedurknoten selbst keine Ausgänge. Es ist auch nicht möglich, eine Ausgangskante eines anderen Knotens mit einem Prozedurknoten zu verbinden.

Um die Flexibilität von Prozedurknoten zu erhöhen, besteht zusätzlich die Möglichkeit, für eine Prozedur eine Liste von Parametern zu definieren. Diese können dann bei jedem Aufruf der Prozedur mit aktuellen Argumentwerten belegt werden. Innerhalb der Prozedur verhalten sich Parameter genau wie andere Variablen. Die Liste der Parameter können Sie bearbeiten, indem Sie die Prozedur durch Doppelklick auf den Prozedurknoten öffnen und dann im Menü `Prozedur` den Befehl `Parameter` wählen. Es erscheint nun ein Fenster mit einer Liste der Parameter.

Zusätzlich bieten Prozedurknoten die Möglichkeit Werte an den aufrufenden Dialogkontext zurückzugeben. Um dies zu nutzen, wählen Sie im Menü `Prozedur` den Befehl `Rückgabewerte`. In der Liste lassen sich weitere lokale Variablen anlegen, die nach dem Ausführen einer Prozedur an den aufrufenden Kontext zurückgereicht werden.

10.6.1 Prozeduren aufrufen

Aufruf-Knoten dienen zum Aufruf von Prozeduren. Mit ihrer Hilfe kann eine Prozedur von verschiedenen Stellen im Dialog aus - ggf. mit unterschiedlichen Argumenten - angesprungen werden. Die Parameter des Aufruf-Knotens sind zum einen die aufzurufende Prozedur sowie die beim Aufruf an die Prozedur zu übergebenden Argumente und die zu verarbeitenden Rückgabewerte. Jedes Argument kann ein komplexer Ausdruck sein, der zum Zeitpunkt des Aufrufs ausgewertet wird. Für die Rückgabewerte können Sie ein beliebiges Muster angeben, auf das der tatsächliche Rückgabewert der Prozedur getestet wird. In der Regel werden Sie hier einfache Variablen-Muster verwenden, um den Rückgabewert in der angegebenen Variablen zu speichern.

Abbildung 10.5: Einstellungen für den Prozeduraufruf

Im obigen Beispiel wird die Prozedur *Prozedur 1* aufgerufen und dabei der Parameter *Anzahl* mit dem Wert *3* initialisiert. Nach Ausführung der Prozedur wird der Wert, der in der Variable *Ergebnis* steht in der Variablen *Summe* gespeichert. Beachten Sie, dass *Ergebnis* eine lokale Variable der Prozedur ist und im Kontext der Aufrufknoten in der Regel *nicht* direkt sichtbar ist. Und umgekehrt ist die Variable *Summe* des aufrufenden Kontexts in der Regel *nicht* direkt sichtbar für die Prozedur.[1]

[1] Eine Ausnahme sind rekursive Aufrufe, in denen sich der Aufrufknoten innerhalb der Prozedur selbst befindet.

Die Anzahl der Ausgänge eines Aufrufknotens entspricht der Anzahl der *Return*-Knoten der aufzurufenden Prozedur. Die Ausführung eines Aufrufknotens führt zu einem Sprung zum Startknoten der Prozedur, wobei die Parameter der Prozedur mit den konkreten Argumenten initialisiert werden. Je nachdem, in welchem Endknoten die Ausführung der Prozedur endet, wird die Ausführung des übergeordneten Dialogs über die entsprechende Ausgangskante des Aufrufknotens fortgesetzt.

Kapitel 11

Externe Geräte

Über die eingebauten Fähigkeiten hinaus ermöglicht DialogOS die Kommunikation mit beliebig vielen externen *Geräten*. Geräte sind externe Module, mit denen die Dialogsteuerung interagiert, zum Beispiel für Texteingabe und -ausgabe, eine Schnittstelle zu Datenbanken, eine Treibersoftware für eine zu steuernde Maschine, etc. Externe Geräte können also sowohl als Eingabemedium für den Dialog dienen als auch Ausgabekommandos empfangen. Dialogsteuerung und externe Geräte kommunizieren über TCP/IP-Streams miteinander, sie können also auch auf verteilten Systemen laufen.

Die Interaktion mit externen Geräten ist eng an die Skriptsprache von DialogOS gekoppelt. Die Informationen, die Sie hin- und herschicken können sind allesamt Werte der Skriptsprache, haben also einen der in Kapitel 5.1 beschriebenen Datentyp. Wie das externe Gerät einen solchen Wert interpretiert (z.B. eine Zeichenfolge als Kommando oder eine Liste oder Struktur als komplexe Informationseinheit) und welche Antwort es ggf. schickt, ist abhängig von dem externen Gerät. Eine externe Datenbank wird z.B. eine Zeichenkette, die Sie schicken, als SQL-Query interpretieren und Ihnen als Antwort eine Liste mit den gefundenen Datenbankeinträgen schicken; eine CD-Spieler wird eine Zahl z.B. als Nummer des zu spielenden Titels interpretieren und gar keine Antwort schicken; usw.

Abbildung 11.1: Definition der Geräte

Sie erstellen und bearbeiten Geräte über den Menübefehl `Geräte` im Menü `Dialog`. Über die Schaltfläche `Neu` können Sie ein neues Gerät anlegen und dieses durch Doppelklick auf den Gerätenamen bearbeiten.

Abbildung 11.2: Einstellungen für ein Gerät

Jedes Gerät muss einen *Namen* und einen Anschluss (*Connector*) haben. Der Connector beschreibt die Art und Weise, wie die Verbindung zwischen DialogOS und dem Gerät aufgebaut wird. Je nach verwendetem Connector müssen Sie verschiedene Parameter angeben. Die von CLT mitgelieferten Module unterstützen allesamt den Connector *CLT Connector (Rendezvous)*. Dieser Connector benutzt eine spezielle Technologie, um im Netzwerk vollautomatisch nach verfügbaren Modulen zu suchen. Dadurch müssen Sie nicht explizit angeben, auf welchem Rechner das externe Modul läuft. Einzige Voraussetzung ist, dass der Parameter *Service Name* mit dem Namen des externen Moduls übereinstimmt.

Den Namen des Geräts können Sie frei wählen, er dient ausschließlich zur Identifikation innerhalb von DialogOS.

Die Ansteuerung der externen Geräte läuft innerhalb von DialogOS wiederum über eigene Knotentypen, die im Folgenden beschrieben werden.

11.1 *Ausgabe*-Knoten

Ausgabe-Knoten erlauben das Senden von Kommandos an externen Geräte. An einem Ausgabeknoten können gleichzeitig mehrere Ausgaben an alle definierten Geräte geschickt werden. Pro Gerät kann eine Liste von Ausdrücken angegeben werden. Jede Liste wird sequentiell abgearbeitet, indem der jeweilige Ausdruck ausgewertet und das Ergebnis an das angegebene Gerät geschickt wird. Die Abarbeitung der Listen erfolgt für alle Geräte gleichzeitig.

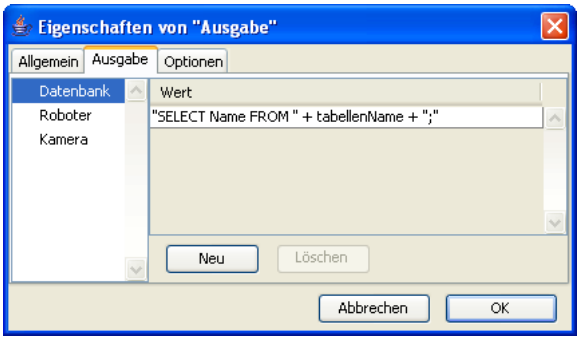

Abbildung 11.3: Einstellungen für Ausgabeknoten

Klicken Sie links auf eines der Geräte, um sich die Ausgabeliste für dieses Gerät anzeigen zu lassen. Klicken Sie auf die Schaltfläche Neu, um einen weiteren Ausdruck zu dieser Liste hinzuzufügen. Durch Doppelklick auf eine Zeile können Sie diese bearbeiten. Die Reihenfolge der Ausdrücke lässt sich durch Klicken und Ziehen der Maus verändern.

Ausgabeknoten haben immer genau eine Ausgangskante, über die der Dialog nach erfolgter Ausgabe fortgesetzt wird. Zusätzlich zu den auszugebenden Werten können Sie einige Optionen für die Ausgabe konfigurieren:

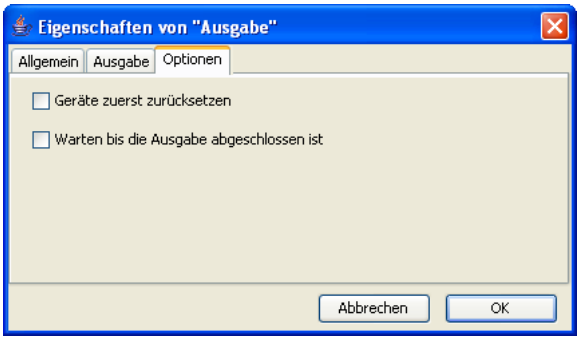

Abbildung 11.4: Weitere Optionen für Ausgabeknoten

Gerät erst zurücksetzen
: Wenn Sie diese Option aktivieren, werden alle noch laufenden Ausgaben (für alle Geräte die dies unterstützen, z.B. ein CD-Spieler der noch im Hintergrund läuft) abgebrochen, bevor die Ausgaben des aktuellen Knotens an die Geräte geschickt werden.

Warten bis die Ausgabe abgeschlossen ist
: Normalerweise wird die Ausführung des Dialogs fortgesetzt, sobald alle Ausgabeelemente an die Geräte geschickt wurden. Dabei kann die eigentliche Ausgabe (Abspielen einer Audiodatei, Vorlesen eines Textes über TTS, Aktion eines Roboters etc.) im Hintergrund noch andauern. Manchmal ist es jedoch notwendig, die tatsächliche Ausgabedauer mit dem Dialogfluss zu synchronisieren. Wenn Sie diese Option aktivieren, wird die Ausführung des Dialogs erst fortgesetzt, wenn alle Geräte zurückmelden, dass die aktuelle Ausgabe komplett abgeschlossen wurde.

11.2 *Eingabe*-Knoten

Eingabe-Knoten dienen zur Verarbeitung von Antworten externer Geräte. Dabei kann es sich z.B. um das Ergebnis eines Spracherkenners handeln, der den Inhalt einer Benutzereingabe an die Dialogsteuerung liefert, oder um eine Datenbank, die auf Anfrage hin eine Liste von Datensätzen zurückliefert.

Jeder *Eingabe*-Knoten ist übrigens ebenfalls *Ausgabe*-Knoten, d.h. Sie können vor der Behandlung der Eingabe Informationen an die Geräte schicken. Dies ist besonders dann interessant, wenn Ausgabe und Eingabe direkt zusammenhängen, wenn also das Gerät direkt auf Ihre Nachricht antwortet.

Wie eingangs beschrieben sind alle Informationen immer elementare Werte der Skriptsprache. Der *Eingabe*-Knoten wartet bei der Ausführung, nachdem er ggf. alle Ausgaben an die Geräte geschickt hat, auf einen Wert von einem externen Gerät und vergleicht diesen dann mit einer Liste von Mustern.

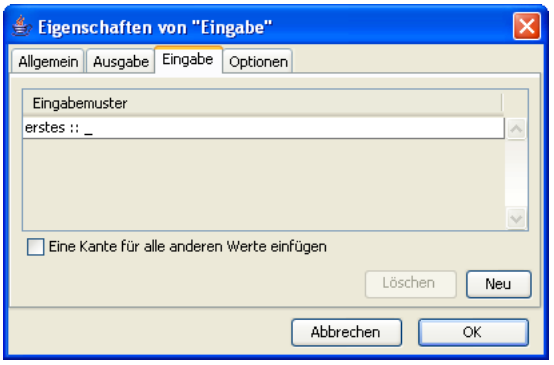

Abbildung 11.5: Einstellungen für Eingabeknoten

Im obigen Beispiel erwartet der *Eingabe*-Knoten zum Beispiel eine Liste und verarbeitet diese mit einem Listenmuster, das den Wert des ersten Listenelements in die Variable *erstes* speichert und den Rest der Liste ignoriert.

Für jedes Muster erhält der Eingabeknoten eine Ausgangskante. Zur Laufzeit wird nun die tatsächliche Eingabe mit den angegebenen Mustern verglichen, und zwar in der Reihenfolge, in der die Muster angegeben sind. Das erste Muster, das auf die Eingabe passt, wird ausgewählt und der Dialog über die dazugehörige Ausgangskante fortgesetzt. Der Dialog verzweigt sich an diesem Knoten also in Abhängigkeit von der Eingabe. Gleichzeitig können Sie durch Variablenbindungen den Wert der Eingabe oder von Eingabeteilen in Variablen speichern, um zu späterem Zeitpunkt darauf zugreifen zu können.

Sollte zur Laufzeit keines der angegebenen Muster auf eine Eingabe passen, so wird diese Eingabe ignoriert, und die Ausführung wartet auf die nächste Eingabe. Um dies zu verhindern, können Sie als letztes Muster der Liste das anonyme Variablenmuster `_` angeben. Dieses passt immer auf alle Eingaben, so dass Sie eine Ausgangskante für alle Eingaben erhalten, die Sie nicht vorher explizit erkannt haben. Alternativ können Sie explizit die Option *Eine Kante für alle anderen Werte einfügen* auswählen.

Die Bearbeitung der Muster-Liste funktioniert analog zu den Ausgabeknoten: Mit den Schaltflächen `Neu` und `Löschen` können Sie Elemente der Liste erzeugen bzw. entfernen. Per Doppelklick können Sie die Muster bearbeiten und per Klicken und Ziehen die Reihenfolge verändern.

Eingabe-Knoten enthalten neben den Optionen für die Ausgabe die folgenden zusätzlichen Einstellungsmöglichkeiten:

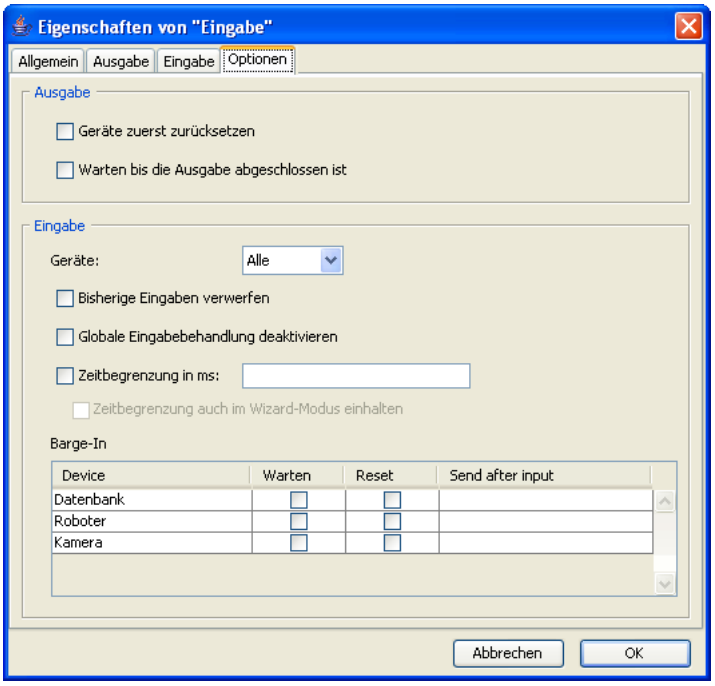

Abbildung 11.6: Weitere Optionen für Eingabeknoten

Gerät
Normalerweise werden an Eingabeknoten die Eingaben aller Geräte berücksichtigt. Sie können jedoch auch angeben, dass an diesem Knoten nur die Eingabe eines einzigen Geräts bearbeitet werden soll.

Bisherige Eingaben verwerfen
Es ist durchaus möglich, dass ein externes Gerät Eingaben an die Dialogsteuerung schickt, während diese sich nicht gerade an einem Eingabeknoten befindet. Diese Eingaben werden in einer Schlange gespeichert, so dass keine Eingabe verloren geht. Bei der Ausführung des nächsten Eingabeknotens wird dann das erste Element dieser Schlange (also die älteste Eingabe) extrahiert und bearbeitet. Dies kann in Einzelfällen nicht erwünscht sein. Aktivieren Sie diese Option, um am Anfang des Eingabeknotens die Eingabeschlange zu leeren und auf eine neue Eingabe zu warten.

Globale Eingabebehandlung deaktivieren
siehe Kapitel 11.3

Zeitbegrenzung
Mithilfe dieser Option können Sie eine zusätzliche Ausgangskante einfügen, über die der Dialog fortgesetzt wird, wenn nach einer bestimmten Zahl von Millisekunden keine Eingabe eingetroffen ist. Dadurch können Sie verhindern, dass der Dialog beliebig lange auf eine Eingabe wartet. Die Länge der Zeitbegrenzung kann durch einen komplexen Ausdruck angegeben werden, der bei jeder Ausführung des Knotens ausgewertet wird. Dadurch können Sie die Zeitbegrenzung in Abhängigkeit vom Dialog variieren.

Barge-In
Diese Option ist nur interessant, wenn Sie Spracheingabe und Sprachausgabe durch externe Geräte realisieren und nicht die eingebauten Komponenten von DialogOS verwenden.

Barge-In bezeichnet die Möglichkeit für den Dialogbenutzer, eine Eingabe zu tätigen, noch während die Ausgabe bearbeitet wird, im Sinne von gesprochener Sprache also dem System ins Wort zu fallen. Um dies zu realisieren ist es zum einen notwendig, dass der Spracherkenner bereits aktiviert wird, während die Ausgabe noch läuft, und zum anderen, dass die laufende Ausgabe abgebrochen wird, sobald eine Eingabe erkannt wurde. Zum anderen stellt sich das Problem, dass man dem Benutzer z.B. ein Zeitfenster von einigen Sekunden zum antworten lassen möchte, dass aber nicht klar ist, wie lange es dauert, die Systemausgabe abzuspielen, sprich ab wann dieses Zeitfenster gemessen werden sollte.

Dieser komplexen Problematik widmet sich die Option "Barge-In". Eine angegebene Zeitbegrenzung läuft erst ab dem Moment, wo alle Geräte, für die die Option "Warten" aktiviert wurde (z.B. die Sprachausgabe), gemeldet haben, dass ihre Ausgabe komplett abgearbeitet wurde. Sobald eine Eingabe eintrifft oder die Zeitbeschränkung abgelaufen ist, kann dann an ausgewählte Geräte (z.B. die Sprachausgabe) ein Signal geschickt werden, alle noch laufenden Ausgaben abzubrechen ("Reset"). Zusätzlich können Sie auch explizit eine Nachricht angeben, die an das Gerät geschickt werden soll (z.B. ein `"stop"`-Kommando für den Spracherkenner).

11.3 Globale Eingabehandlung

DialogOS bietet die Möglichkeit, Eingabemuster, die an *allen* Eingabeknoten eines (Teil-)Dialogs vorgesehen sind, kompakt in einer so genannten globalen Eingabebehandlung unterzubringen. Eine globale Eingabebehandlung besteht aus einem Eingabemuster und einem Untergraphen, der dann durchlaufen wird, wenn das Muster auf die Eingabe passt. Dies hat den Vorteil, dass häufig verwendete Muster und ihre nachfolgende Dialoglogik nicht an jedem einzelnen Eingabeknoten hinzugefügt werden müssen, sondern einmal zentral definiert werden können.

Um globale Eingabemuster zu erstellen und zu verwalten, wählen Sie den Menübefehl `Globale Eingabebehandlung` im Menü `Graph`. In dem nun erscheinenden Fenster erhalten Sie eine Liste von bestehenden Eingabebehandlungen, also ihren Namen, dem jeweiligen Muster und Typen.

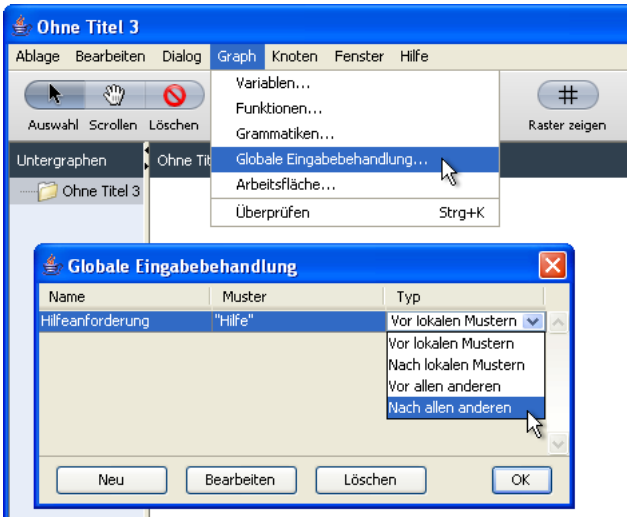

Abbildung 11.7: Globale Eingabebehandlung

Der Name einer Eingabebehandlung dient nur zur besseren Orientierung für den Dialogentwickler. Der Typ der Eingabebehandlung macht eine Aussage darüber, an welcher Stelle in einem Eingabeknoten das

globale Eingabemuster in die Reihenfolge der lokalen Muster eingefügt wird. Es gibt vier Typen, die aus einer Liste ausgewählt werden können:

Vor allen anderen
Die Eingabebehandlung wird als erstes, vor allen anderen Mustern, geprüft.

Nach allen anderen
Die Eingabebehandlung wird als letztes, nach allen anderen Mustern, geprüft.

Vor lokalen Mustern
Die Eingabebehandlung wird *vor* den lokalen Mustern des jeweiligen Eingabeknotens geprüft, aber *nach* den globalen Eingabebehandlungen des übergeordneten Graphen. Diese Option wirkt sich nur aus, wenn Sie in globale Eingabebehandlungen in Untergraphen oder Prozeduren definieren.

Nach lokalen Mustern
Die Eingabebehandlung wird *direkt nach* den lokalen Mustern des jeweiligen Eingabeknotens geprüft, also noch *vor* den globalen Eingabebehandlungen des übergeordneten Graphen. Diese Option wirkt sich nur aus, wenn Sie in globale Eingabebehandlungen in Untergraphen oder Prozeduren definieren.

Die Schaltflächen guibuttonHinzufügen, `Entfernen` und `Bearbeiten` bieten die Möglichkeit, neue Eingabemuster zu erstellen und bestehende zu entfernen oder zu bearbeiten.

Sie können den Namen und das Muster der Eingabebehandlung durch Doppelklick auf das jeweilige Feld direkt bearbeiten. Wenn Sie die Schaltfläche `Bearbeiten` anklicken, erscheint der Untergraph, der ausgeführt wird, wenn eine Eingabe auf das Muster des Eingabehandlers gepasst hat. Dieser Teildialog verhält sich so wie der eines Untergraphen. Er unterscheidet sich lediglich durch die verfügbaren Endknoten.

11.3.1 *Wiederholen*-Knoten

Wenn Sie den Untergraph der Eingabebehandlung mit einem *Wiederholen*-Knoten beenden, wirkt sich dies wie eine Schleife aus. Die Dialogführung springt zurück zum Eingabeknoten, an dem die Eingabebehandlung ausgelöst wurde und führt diesen erneut aus. Beispiel für eine solche Anwendung wäre eine Eingabebehandlung, die auf die Benutzereingabe `"Wie␣bitte"` die zuletzt abgespielte Systemausgabe noch einmal ausgibt und dann die Ausführung des Eingabeknotens wiederholt.

11.3.2 *Fortfahren*-Knoten

Wenn Sie den Untergraph der Eingabebehandlung mit einem *Fortfahren*-Knoten beenden, wird für diesen an *jedem* Eingabeknoten, für den die Eingabebehandlung gilt, automatisch ein Ausgang angelegt, über den der Dialog fortgeführt wird. Beispiel für eine solche Anwendung wäre eine Eingabebehandlung, die auf eine Benutzereingabe eine einheitliche Prozedur startet, nach der dann aber lokal am Eingabeknoten entschieden werden soll, wie es weitergeht.

Wenn Ihre Eingabebehandlung mehrere *Fortfahren*-Knoten enthält, werden entsprechend mehrere Ausgangskanten angelegt, so dass Sie unterscheiden können, in welchem Knoten die Ausführung endete.

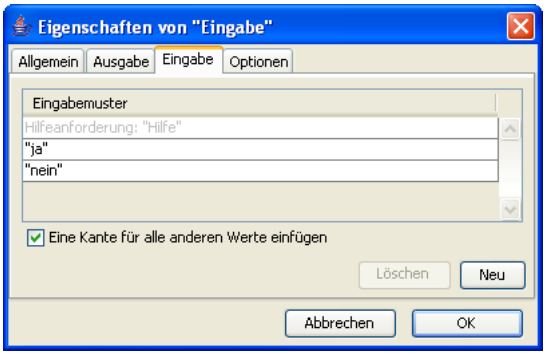

Abbildung 11.8: Eingabeknoten mit globaler Eingabebehandlung

Wenn Sie globale Eingabebehandlungen definiert haben, sehen Sie diese auch direkt, wenn Sie die Einstellungen eines *Eingabe*-Knotens bearbeiten. Sie können für einen einzelnen *Eingabe*-Knoten die globale Eingabebehandlung deaktivieren, indem Sie unter dem Karteireiter `Optionen` die gleichnamige Option auswählen.

Anhang A

Hilfsfunktionen in DialogOS

DialogOS stellt folgende eingebaute Funktionen zur Verfügung, die innerhalb von Ausdrücken und Skripten verwenden werden können. Der Datentyp α zeigt dabei an, dass es sich um einen beliebigen Typ handeln kann. Bei Funktionen mit Rückgabetyp α hängt der Typ des Rückgabewerts in der Regel von den konkreten Funktionsargumenten ab.

int abs(int i)
Gibt den absoluten Wert einer Zahl i zurück.

struct addDate(struct date, struct addition)
Berechnet die Summe aus date und addition. date muss ein volle Datumsstruktur wie folgt sein: day:int, month:int, year:int, addition muss nicht alle Datenfelder haben. Jahre werden zuerst addiert, danach Monate und dann Tage.

Beispiele:
`{ day=10, month=6, year=2003 } + { day=25 }` = `{ day=5, month=7, year=2003 }`
`{ day=10, month=6, year=2003 } + { day=25, month=1 }` = `{ day=4, month=8, year=2003 }`

string charAt(string s, int index)
Gibt den Buchstaben an der Stelle index in einem String s zurück.

list append(list l1, list l2)
Gibt eine neue Liste aus l1 und l2 konkateniert zurück.

list concat(list lists)
Gibt eine neue Liste bestehend aus allen Listen in lists konkateniert zurück.

list cons(α v, list l)
Gibt eine neue Liste zurück, in der v vor alle Elemente aus l gestellt ist.

bool contains(list list, α v)
Gibt zurück, ob v in list enthalten ist.

bool contains(string s, string infix)
Gibt zurück, ob infix in s enthalten ist.

bool contains(struct str, string key)
Gibt zurück, ob eine Struktur str das Label key beinhaltet.

struct currentDate()
Gibt die momentane Uhrzeit als Struktur der Form day:int, month:int, year:int zurück.

struct currentTime()
Gibt das momentane Datum als Struktur der Form h:int, m:int zurück.

int currentTimeMillis()
Gibt den genauen Zeitabstand zwischen der momentan Uhrzeit und der um Mitternacht am 1. Januar 1970 in Millisekunden zurück.

struct currentWeekday()
Gibt den momentanen Wochentag und die Kalenderwoche als Struktur der Form day:int, week:int zurück. Wochentage beginnen bei 0: 0=Sonntag, 1=Montag, ...

bool empty(list l)
Gibt zurück, ob eine Liste l leer ist.

bool endsWith(string s, string postfix)
Gibt zurück, ob s mit postfix endet.

list enumerate(int start, int end, int interval)
Gibt eine Liste mit allen Zahlen zwischen der von start und end aus, mit einem jeweiligen Abstand von interval.

void error(string e)
Wirft eine Fehlermeldung mit dem Text von e.

real exp(real r)
Gibt die Euler'sche Zahl mit einer Potenz von r zurück.

α **get(list l, int index)**
Gibt das Element an der Stelle index einer Liste l zurück

α **get(struct str, string key)**
Gibt das Element von str mit dem Label key zurück oder undefined, falls key nicht in str enthalten ist.

int indexOf(list list, α **v)**
Gibt die Position von v in list zurück oder -1, falls v nicht in list enthalten ist.

int indexOf(string s, string infix)
Gibt die Position von infix in s zurück oder -1, falls infix nicht in s enthalten ist.

bool isInt(string s)
Gibt zurück, ob s zum Typ int konvertiert werden kann.

bool isReal(string s)
Gibt zurück, ob s zum Typ real konvertiert werden kann.

list keys(struct str)
Gibt eine Liste zurück, die alle Labels von str enthält.

int length(list l)
Gibt die Anzahl der Elemente von l zurück.

int length(string s)
Gibt die Länge eines Strings s zurück.

struct loadProperties(string directory, string filename)
Liest eine Property-Datei mit dem Dateinamen filename aus dem Verzeichnis directory und gibt deren Inhalt als Struktur zurück.

real log(real r)
Gibt den natürlichen Logarithmus einer Zahl r zurück.

struct member(list list, α v)
Äquivalent zu contains(list, v)

struct merge(struct str1, struct str2)
Gibt eine neue Struktur zurück, die sich aus einer Zusammenfügung von str1 und str2 ergibt.

Die Parameter-Strukturen werden nicht abgeändert. Stattdessen wird eine neue Struktur erstellt und zurückgegeben. Wenn str1 und str2 das gleiche Label beinhalten, so bestimmt sich das Ergebnis aus dem Typ der untergeordneten Werte. Sind die Werte von zwei Elementen aus str1 und str2 selbst Strukturen, so werden diese Unterstrukturen rekursiv zusammengefügt. Ansonsten wird der Wert von str2 übernommen.

Beispiele:
`merge({ x=3, y=4 }, { x=4, z=5 })` = `{ x=4, y=4, z=5 }`
`merge({ x={x=3}, y={a=4} }, { x={y=4}, y=5 })` = `{ x={x=3, y=4}, y=5 }`

struct mergeDefined(struct str1, struct str2)
Gibt eine neue Struktur zurück, die sich aus einer Zusammenfügung von str1 und str2 ergibt. Im Gegensatz zu merge werden allerdings nur Werte von str1 übernommen, wenn sie in str2 nicht den Wert undefined haben.

int parseInt(string s)
Konvertiert s zu einem Wert mit dem Typ int. Gibt undefined zurück, falls die Konvertierung fehlschlägt.

real parseReal(string s)
Konvertiert s zu einem Wert mit dem Typ real. Gibt undefined zurück, falls die Konvertierung fehlschlägt.

void print(α v)
Gibt eine String-Repräsentation von v auf der Kommandozeilen-Ebene (stdout) zurück.

void printf(string format, ...)
Gibt eine formatierte Zeichenkette auf der Kommandozeile (stdout) aus. Diese Funktion arbeitet analog zur Funktion printf aus der C Standard Library.

struct put(struct str, string a, α x)
Gibt eine Struktur zurück, die str entspricht und zusätzlich ein Element a mit dem Wert x enthält. Falls die Struktur str a bereits enthält, wird der Wert überschrieben.

int random(int min, int max)
Gibt einen Zufallswert aus dem geschlossenen Interval von min und max zurück.

list remove(list list, list remove)
Gibt eine Liste zurück, die alle Elemente aus list abzüglich derer aus remove enthält.

string replace(string s, string a, string b)
Gibt eine Kopie von s zurück, in der alle Vorkommen von a durch b ersetzt wurden.

list reverse(list list)
Gibt eine Kopie von list in umgedrehter Form zurück.

int round(real r)
Rundet r zum nächstliegenden Wert des Typs int.

void showDialog(...)
Konkateniert die String-Repräsentation der Argumente und erzeugt dann ein Dialogfenster in der Programmoberfläche, das diesen String anzeigt, bis Sie das Fenster wieder schließen. Diese Funktion ist hauptsächlich zu Testzwecken zu gebrauchen.

list sort(list list)
Gibt eine sortierte Liste mit dem gleichen Inhalt wie die Liste list zurück. Alle Elemente von list müssen dabei vom gleichen Typ sein.

bool startsWith(string s, string prefix)
Gibt zurück, ob der String s mit prefix beginnt.

string str(α s)
Konvertiert ein beliebiges Element s zu einem String.

list stripSpecialCharacters(list list)
Nimmt als Input eine Liste list von Strings und gibt eine Liste zurück, die die gleichen Elemente enthält, allerdings ohne Zeichen, die keine Buchstaben oder Zahlen sind.

α head(list l)
Gibt das erste Element (den Kopf) einer Liste l zurück.

list tail(list l)
Gibt den "Rest"(tail) einer Liste l zurück, der entsteht, wenn das erste Element weg gelassen wird.

list sublist(list list, int offset)
Gibt die Teilleiste von list zurück, die an der Stelle offset beginnt.

list sublist(list list, int offset, int length)
Gibt diejenige Teilliste von list zurück, die an der Stelle offset beginnt und die Länge length hat.

string substring(string s, int offset)
Gibt den Teilstring von s zurück, der an der Position offset beginnt.

string substring(string s, int offset, int length)
Gibt denjenigen Teilstring von s zurück, der an einer Position offset beginnt und die Länge length hat.

string toLowerCase(string s)
Gibt eine Kopie von s zurück, in der alle Buchstaben zu Kleinbuchstaben konvertiert wurden.

string toUpperCase(string s)
Gibt eine Kopie von s zurück, in der alle Buchstaben zu Großbuchstaben konvertiert wurden.

list values(struct str)
Gibt eine Liste mit allen Werten aus str zurück.

Index