RoboLang oder so (mir ist noch immer nichts besseres eingefallen)

Roland Schweiger (Projektleiter)
Gerald Stieglbauer

4. April 2000

Zusammenfassung

Dieses Dokument beschreibt die Entwicklung einer graphischen Programmiersprache samt zugehöriger Programmierumgebung (IDE). Die Sprache ist speziell auf die Programmierung von Robotern abgestimmt. Es wird weiters eine Schnittstelle zu dem am Institut für Computerwissenschaften der Universität Salzburg entwickelten Robotersimulator DIMAS, sowie dem Robotor EMMA implementiert.

Verwendet wird die Programmiersprache JAVA und das Entwicklungstool <NetBeans/Forte for Java/JBuilder>. Als Output des graphischen Programms soll wiederum ein JAVA-Programm entstehen.

Inhaltsverzeichnis

Kapitel 1

Planungsphase

Hier werden folgende Punkte genauer behandelt:

- Welche Elemente werden in der graphischen Programmiersprache (GPS) verwendet?
- Was sind die genauen Funktionen und Eigenschaften dieser Elemente?
- Wie werden diese Elemente in die Programmierumgebung (IDE) eingebettet?
- Wie stellt sich der genaue Ablauf eines fertigen Programmes dar?
- Wie sehen die genauen Schnittstellen zu DIMAS und EMMA aus?

Dieses Kapitel versteht sich als abstrakte Festlegung von Grundkonzepten des Projektes und soll bereits eine innere Struktur der zu programmierenden Anwendungssoftware diskutieren. Wie groß das „Fenster“ zum Anwender der Software dann tatsächlich ist sowie die wichtigen Punkte wie Ergonomie, Benutzerfreundlichkeit, Zusatzprogramme, genaue Festlegung der Bedienelemente der IDE etc. wird im Kapitel 2 (Entwurfsphase) behandelt.

1.1 Elemente der GPS

Prizipiell lassen sich zwei Grundelemente unterscheiden:

- Units
- Connections (Verbindungen) zwischen den Units

1.1.1 Units

Eine Unit besteht also aus folgenden *Unit-Attribute* (siehe Abb. 1.1):

- Objekttyp: Legt die allgemeine qualitative Eigenschaft der Unit fest
- Funktionstyp: Bestimmt die funktionelle Eigenschaft (Bezeichnung der Übertragungsfunktion) der Unit

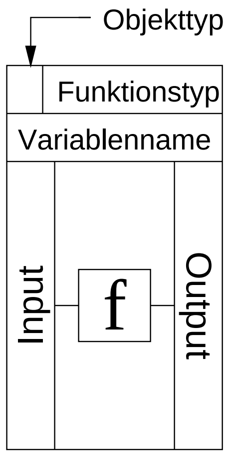

Abbildung 1.1: Schematische Darstellung einer Unit

- Variablenname einer Instanz einer speziellen Unit
- Eingangsparameter (Inputs)
- Ausgangsparameter (Outputs)
- Eingehende und ausgehende Steuerleitungen (optional und nicht in Abb. 1.1 dargestellt)
- Übertragungsfunktion
- Optionenfenster (da nicht einheitlich ebenfalls nicht in Abb. 1.1 dargestellt)
- ReihungsID (nicht in Abb. **??** dargestellt)

Objektyp: Unter allgemeine qualitative Eigenschaften versteht man die Art und Weise, wie eine Unit im Inneren aufgebaut ist. Prinzipiell wird zwischen folgenden Eigenschaften unterschieden:

- Roboter-Komponenten (Component-Units)
- JAVA-Programme (Java-Units)
- Fuzzy-Regler (Fuzzy-Units)
- Neuronale Netze (Net-Units)
- Super-Units
- Special-Units (Black-Boxes mit spezifisch festgelegter Übertragungsfunktion wie z.B. Kontroll-Units siehe 1.6)

Die ersten vier Punkte sind jeweils Spezialfälle des letzten Punktes. Units vom Typ Roboter-Komponenten sind Units, die aufgrund eines vorgegebenen Roboters in einer speziellen Roboterbibliotek zur Verfügung gestellt werden (wie z.B Roboterkamera, Motoren, etc).

Eine Super-Unit ist eine neue Unit, dessen innere Struktur aus mehreren verschalteten Units aufgebaut ist. Special-Units sind Units mit sehr speziellen, hier nicht weiter spezifizierten Eigenschaften, unter denen weitere Spezialfälle zusammengefaßt werden.

Funktionstyp: Unter einem Funktionstyp werden solche Units unter einem spezifischen Namen zusammengefaßt, deren Objekttypeigenschaften bereits vollständig definiert sind (z.B. ein fertiges JAVA-Programm, ein ausformulierter Fuzzy-Regler, ein fertig konstruiertes neuronales Netz etc.). In der objektorientierten Programmierung (OOP) würde das entsprechende Äquivalent dazu ein fertig programmiertes Objekt sein, von dem man nun mehrere Instanzen erzeugen kann.

Solche fertige Units können entweder vom Programmierer erstellt werden, oder können in Bibliotheken zur Verfügung gestellt werden und mit teilweise beschränktem Zugriff vom Programmierer modifiziert werden.

Eine Ausnahme stellen dabei die Component-Units dar. Von ihnen kann im gesamten Programm auf allen Ebenen nur eine Instanz erzeugt werden. Anschließend kann das spezifische Bauteil nicht mehr aus der Roboterbibliothek ausgewählt werden. Auch hat der Programmierer mit Ausnahme der Festlegung von Bauteil-Optionen (wie z.B. Auflösung einer Kamera etc.) keinen Zugriff auf die innere Struktur. Weiters ist der Name des Funktionstyps bereits festgelegt und kann nicht verändert werden.

Eine Ausnahme von dieser Ausnahme sind jedoch jene Component-Units, die nur Ausgänge, aber keine Eingänge besitzen. Bei dieser Art von Units ist es möglich mehrere Instanzen zu erzeugen, sonst werden sie aber wie normale Roboterbauteil-Units behandelt.

Variablenname: Ein Bezeichner der eine jeweilige Instanz einer Unit in der jeweils aktuellen Programmierebene eindeutig identifiziert.

Ausnahme hier wiederum die Component-Units: Deren Variablennamen sind bereit festgelegt und können nicht mehr verändert werden. Da i. a. jeweils nur eine Instanz in allen Programmierebenen erzeugt werden kann, ist auch der Variablenname in allen Programmierebenen eindeutig.

Eingangs/Ausgangsparameter, Übertragungsfunktion: Jede Unit besitzt eine variable Anzahl (darf auch null sein) von Eingangs- und Ausgangsparametern (oder Input- bzw. Outputpins). Die Ausgangparameter werden durch die Übertragungsfunktion anhand der Eingansparameter berechnet.

Jeder Eingangs- und jeder Ausgangsparameter hat einen bestimmten *Typ* (sh. 1.2). Ein einzelner Typ können sich auch aus mehreren Typen zusammensetzten.

Weiters besitzen die Eingangs- und Ausgangsparameter *Bezeichner*. Diese sind standartmäßig auf bis `i0` bis `ix` bzw. `o0` bis `ox` gesetzt, könner aber vom Programmierer unbenannt werden.

ReihungsID: Legt die bevorzugte Reihenfolge fest, in der die einzelnen Ausgangsparameter der Units anhand der Eingangsparameter berechnet werden (sh. 1.5.1).

1.1.2 Connections

Verbindungen (Connections) können von einzelnen Ausgangsparameter zu Eingangsparametern *gleichen* Typs gezogen werden. Dabei handelt sich um gerichtete Verbindungen, durch die Daten an weitere Units weitergeleitet werden. Auf einer solchen Connection befindet sich ein einelementiger Datenspeicher der genau einen Datenwert vom Typ der Connections, der durch die vor- und nachgeschalteten Units bestimmt wird, aufnehmen kann.

Ändert sich der Datentyp eines Parameters einer Unit, so wird die Verbindung als ungültig markiert (aber nicht gelöscht), solange bis wieder eine Typkonsistenz gegeben ist.

Weiters besitzen Connections auch einen im jeweiligen Frame eindeutigen Bezeichner. Standartmäßig stimmt dieser mit dem Bezeichner des korrespodierenden Output-Pins überein. Handelt es sich um eine Connection die an einem Input-Feld angeschlossen wurde, so wird der Bezeichner des Pins des Input-Feldes dafür herangezogen. Der Programmierer kann diesen Bezeichner jedoch jederzeit neu benennen, solange dieser unter allen im Frame gezogenen Connections eindeutig bleibt.

1.2 Datentypen:

Jedem Eingangs- und Ausgangsparameter einer Unit muß ein Datentyp zugewiesen werden. Ein Datentyp kann sein:

- Atomarer Datentyp
- Zusammengesetzter Datentyp (Struktur)
- Ein- oder mehrdimensionale Arrays (Datenfeld)
- Datentypen vom Typ `intern`

Eine Gleichheit zwischen zwei Datentypen ist nur dann vorhanden, wenn es sich tatsächlich um eine identische Datentypdefinition handelt. Eine Umwandlung von atomaren Datentypen muß explizit durch eine zwischengeschaltet Unit erfolgen. Auch besteht bei zusammengesetzten Datentypen keine Gleichheit, wenn diese intern gleich deklariert wurden, jedoch nach außen verschiedene Namen aufweisen.

Eine auf allen Programmierebenen gleiche Datentypbibliothek soll die Datentypen verwalten und kann vom Programmierer erweitert werden. Atomare und vom System vorgegebene Datentypen können von dieser Datenbank weder gelöscht noch modifiziert werden.

1.2.1 Atomare Datentypen:

Es wird zwischen folgenden atomaren Datentypen unterschieden:

- `byte` (8 Bit)
- `short` (16 Bit)
- `int` (32 Bit)

- `long` (64 Bit)
- `float` (32 Bit)
- `double` (64 Bit)
- `char` (16 Bit)
- `boolean` (1 Bit)

1.2.2 Zusammengesetzte Datentypen oder Strukturen

Es sollen auch sogenannte Strukturen festgelegt werden können. Unter einer Struktur wird ein neuer Datentyp zusammengefaßt, der wiederum aus gültigen Datentypen besteht. Ein gültiger Datentyp ist rekursiv definiert als entweder ein atomarer Datentyp oder eine weitere Struktur.

In Bibliotheken sollen wiederum solche zusammengesetzte Datentypen dem Programmierer zur Verfügung gestellt werden. Weiters soll der Programmierer eigene Datentypen entwerfen und in Bibliotheken verwalten können. Dafür soll ein eigener Datentypeditor dem Programmierer als Hilfe gegeben werden.

1.2.3 Ein- oder mehrdimensionale Arrays

Natürlich können auch Datenfelder verwendet werden. Wesentlich dabei ist, daß die Größe und Dimension eines Arrays von Anfang an festgelegt wird. Datentypgleichheit besteht nur dann, wenn zwei Arrays den gleichen Datentyp für jedes Arrayelement verwenden und dieselbe Größe und Dimension aufweisen.

1.2.4 Interne Datentypen

Darunter fallen Steuerleitungen (Connections vom Type `intern`) die sich auf die interne Struktur der Programmiersprache beziehen. Denkbar in diesem Zusammenhand wären zum Beispiel Steuerleitungen, die die internen Parameter von Membershipfunctions von Fuzzy-Reglern aktiv während des Programmablaufes verändern. Weiters sollen solche Datentypen bei sogenannten Kontroll-Strukturen eine wesentliche Rolle spielen (siehe 1.6). Es ist noch zu überlegen, ob und wie diese Datentypen eventuell in die oben genannten Typen konzeptuell eingegliedert werden könnten.

1.3 Konstanten und Initialwerte

Konstante und Initialwerte sind bei der hier zu entwerfenden graphischen Programmiersprache eng verwandte Begriffe.

Manchmal wird es notwendig sein Units Initialwerte zu geben, um mit Berechnungen beginnen zu können (so z.B. bei rekursiven Verbindungen). Deshalb kann man jedem Eingangsparameter einer Unit einen Initialwert zuweisen. Dieser muß natürlich mit dem Datentyp übereinstimmen. Für komplexere Datentypen, wäre es hier wierderum sinnvoll eine Datenbank mit für den jeweiligen Datentyp gültige Initialwerte zur Verfügung zu stellen, die wiederum vom Programmierer erweitert werden könnte.

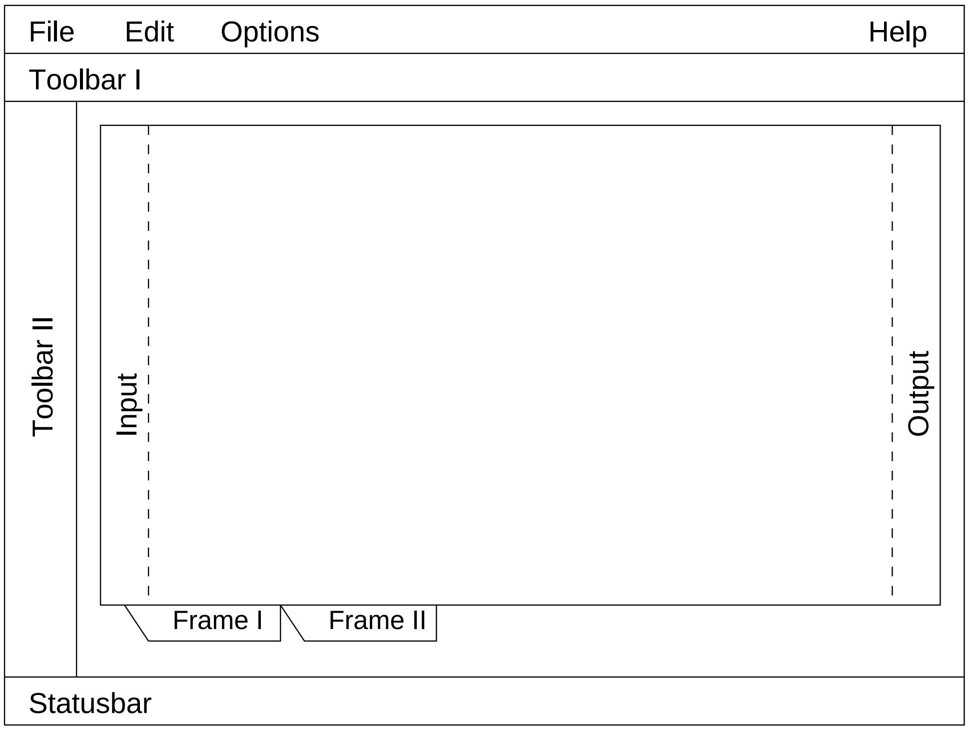

Abbildung 1.2: Schematischer Entwurf der IDE

Der Initialwert wird, falls eine Connection am Eingangsparameter anliegt genau einmal eingelesen. Falls keine Connection anliegt so wird der Initialwert als Konstante verwendet, die bei jedem Berechnungsschritt anliegt.

1.4 Einbindung der Elemente in die IDE

Die IDE besteht prizipiell aus folgenden Elementen (siehe Abb. 1.2):

- Pull-Down Menüs: Stellt den gesamten Befehlssatz der IDE dar.
- Einer horizontalen Toolbar, in der die wichtigsten Elemente der Pull-Down Menüs vorkommen.
- Einer vertikalen Toolbar zur Auswahl und Manipulation von GPS-Elementen.
- Einer Statusleiste, zum Anzeigen von Hilfen und sonstigen Meldungen.
- Einer Framebereich, in der die Elemente der GPS plaziert werden können. Dieser besitzt weiters
  - eine Input-Leiste und
  - eine Output-Leiste.

1.4.1 Plazieren von Elementen

GPS-Elemente können nun in der vertikalen Toolbar aus einer Datenbank ausgewählt oder gänzlich neu erzeugt werden. Beim Auswählen aus einer Datenbank soll ein Libary-Manager aufgerufen werden, aus der der Programmierer bequem seine Elemente auswählen kann. Beim Erstellen von neuen Elementen muß der Programmieren zwischen den einzelnen Unit-Typen auswählen.

Danach kann das ausgewählte Element auf einer Frame plaziert (instanziert) werden. Durch Anklicken des Elementes können seine spezifischen Eigenschaften je nach Unit-Typ eingestellt oder programmiert werden.

Anschließend können Connections zwischen den Unit gezogen werden. Dies soll ebenfalls in der vertikalen Toolbar anwählbar sein.

Außerdem sollen in der vertikalen Toolbar noch diverse Buttons zur Manipulation der Anordnung der Elemente vorhanden sein (wie z.B. Löschen, Verschieben, Kopieren, etc.)

1.4.2 Programmierebenen

Bei der Verwendung von Super-Units muß zwischen verschiedenen Programmierebenen unterschieden werden:

- Mainframe
- Subframe
- Special-Frame

Mainframe: Das Mainframe stellt das Hauptprogramm dar. Bei kleineren Programmen wird man sich darauf beschränken können, nur eine Mainframe und keine Subframe zu verwenden.

Es ist aber auch erlaubt zur Verbesserung der allgemeinen Übersicht mehrere Mainframes zu verwenden (natürlich unter Überprüfung der Component-Unit-Abhängigkeiten wie unten in Subframe beschrieben). Es handelt sich dann dabei um zwei völlig getrennte Pfade, die in weiterer Folge als unabhängige Threads implementiert werden könnten, die aber auch keine direkte Möglichkeit zur Kommunikation besitzen.

Das Hauptprogramm kann diverse Aufrufparameter benötigen. Dies wird durch Ziehen von Connections von im Mainframe plazierten Units auf die linke Input-Leiste (sh. 1.3) des Frames ermöglicht. Weiters können beim Beenden des Programmes Rückgabewerte an den aufrufenden Prozeß übergeben werden. Dies geschieht analog durch ziehen von Connections an die rechte Output-Leiste.

Subframe: Will man nun zur übersichtlicheren Strukturierung eines Programms mehrere, in einem funktionellen Zusammenhang stehende Units zu einer einzigen Unit (Super-Unit) zusammenfassen, so ist die Verwendung von Subframes notwendig. Dieses unterscheidet sich hauptsächlich dadurch von einem Mainframe, da es nicht als eigenständiges Programm gestartet werden kann.

Ein Subframe definiert also eine neue Super-Unit und kann daher als Unit in einer User-Libary verwaltet werden. Somit kann dieses erzeugte Element in der Mainframe als eigenständige Unit eingesetzt werden. Weiters kann in einem Subframe auch auf andere bereits definierte Super-Units zugegriffen werden.

Die Eingangs- und Ausgangsparameter des neuen Unit-Blocks werden durch Ziehen von Connections im Subframe-Fenster nach der Input- und Outputleiste definiert. Somit werden dort auch die Datentypen festgelegt.

Einschränkungen sind bei der Verwendung von Component-Units zu beachten. Prinzipiell ist allgemein zu unterscheiden:

- Roboterspezifische Super-Units
- Roboterunabhängige Super-Units

Roboterspezifische Super-Units können nur ausgewählt werden, wenn alle in den Subunits vorkommenden Roboterbauteil-Units mit dem aktuell in Verwendung befindlichen Robotor übereinstimmen und diese auch noch nicht in irgendeiner, dem aktuellen Projekt zugehöriger Unit verwendet werden, da i. a. jede Component-Unit im gesamten Projekt nur ein mal verwendet werden darf.

Eine Ausnahme stellen, wie oben beschrieben jene Component-Units dar, die nur Ausgänge, aber keinerlei Eingänge besitzen. Diese können beliebig oft instanziert werden. Es gilt jedoch die Einschränkung, daß bei einer Veränderung von Bauteiloptionen einer solchen Unit sich diese Optionsänderung bei allen Instanzen gleichzeitig auswirkt.

Roboterunbahängige Super-Units enthalten keine Roboterbauteil-Units und können daher beliebig oft instanziert werden.

Es ist zu überlegen, wie oben genannte Abhängigkeiten für den Programmierer übersichtlich dargestellt werden könnten.

Special-Frame: Diese dritte Art von Frame faßt all jene Fenster zusammen, mit denen sich manche Unit-Typen näher spezialisieren lassen. Diese Frames sind jedoch klar von den Optionenfenstern zu unterscheiden. Während in den Optionenfenstern u.a. einige Eigenschaften der Übertragungsfunktion eingestellt werden können, wird in einem Specialframe die gesamte Übertragungsfunktion definiert.

Zudem werden die Buttons in der vertikalen Toolbar bei Auswahl eines derartigen Frames, den neuen Bedürfnissen angepaßt. D.h. wenn es sich z.B. um einen Editor handelt, so werden Buttons wie Einfügen, Kopieren, Ausschneiden, Löschen etc. zur Verfügung gestellt.

Sonst werden Special-Frames wie Subframes behandelt, da auch sie eine neue Unit definieren.

Special-Frames sind derzeit nur bei der Neuprogrammierung von Units vom Typ Java und Fuzzy vorgesehen.

Projekt: Zu einem Projekt gehören alle neu programmierten Frames (sowohl Mainframe als auch Subframe). Es können weiters Units und Datentypen aus anderen Projekten importiert werden, sofern sie in den entsprechenden Datenbanken eingetragen wurden; auf deren Source kann jedoch aus leicht einzusehenden Gründen nicht zugegriffen werden. Weiters sind bei der Verwendung von Super-Units die oben genannten Einschränkungen zu beachten.

Weiters kann ein Projekt unter Berücksichtigung aller verwendetet Roboterbauteil-Units ein anderes Projekt als Super-Unit einbinden.

Es wäre weiters eine Einführung eines Übersichtbaumes bei großen Projekten sinnvoll, um eine klare Baumstruktur aller im Projekt programmierten Frames (aller Typen) darzustellen.

1.5 Runtime-Strategien:

Die Punkt umfaßt einen Lösungsansatz zu folgende Fragen:

- Wie wird nun das fertige Projekt zur Laufzeit abgearbeitet?
- In welcher Reihenfolge werden die Unit zur Berechnung der Output-Daten anhand der Input-Daten und der Übertragungsfunktion herangezogen?
- Wie funktioniert die Datenübergabe von einer Unit zur nächsten?
- Wie werden Super-Units abgearbeitet?

1.5.1 Vergabe von ReihungsIDs

Die ReihungsID soll eine Richtlinie angeben, in welcher Reihenfolge die einzelnen Units ausgeführt werden sollen.

Zu erst wenden wir uns dem einfachsten Fall zu: Es existiert nur ein Frame - das Mainframe. Es werden weiters keine Super-Units verwendet. Außerdem gibt es keine Verzweigungen, Rückkopplungen oder sonstige Schleifen. Ein Beispielprogramm ist in Abb. 1.3 dargestellt.

In welcher Reihenfolge werden nun die Units abgearbeitet? Es sind folgende zwei Grundstrategien denkbar:

- Automatische Vergabe
- Manuelle Vergabe

Für jedes Frame soll nun vom Programmierer entschieden werden welche Strategie angewendet werden soll. Bei der manuellen Vergabe soll ein Unitmanager für das jeweilige Frame den Programmierer dabei unterstützen. In diesem Unitmanager besteht also die Möglichkeit die zu bevorzugende Abarbeitungsreihenfolge zu definieren.

Die manuelle Vergabe soll allerdings nur in Spezialfällen eingesetzt werden (z.B. für eine Zeitoptimierung). Bei der automatischen Vergabe wird nun nach Algorithmus 1 vorgegangen, der auch kompliziertere Fälle wie in Abb. 1.4 behandelt.

Welche Strategie in den speziellen Unterteilungen des Algorithmus 1 jeweils verwendet wird, ist vom Programmierer abhängig und kann durch Optionen in einzelnen Units bzw. der Frames festgelegt werden. Es wird von Anwendungsfall zu Anwendungsfall verschieden sein, welche Strategie die bessere ist.

1.5.2 Parameterübergabe zwischen Units

Zwischen jeden Units, die durch eine Connection miteinander kommunizieren, befindet sich auf jeder Verbindung ein Datenspeicher der genau ein Element des zugehörigen Datentyps aufnehmen kann. Wie man sich das graphisch vorstellen kann ist in Abb.1.5 dargestellt.

Wie zur Laufzeit dann die einzelnen Units ausgewählt werden und die Parameterübergabe genau funktioniert, wird durch Algorithmus 2 bestimmt.

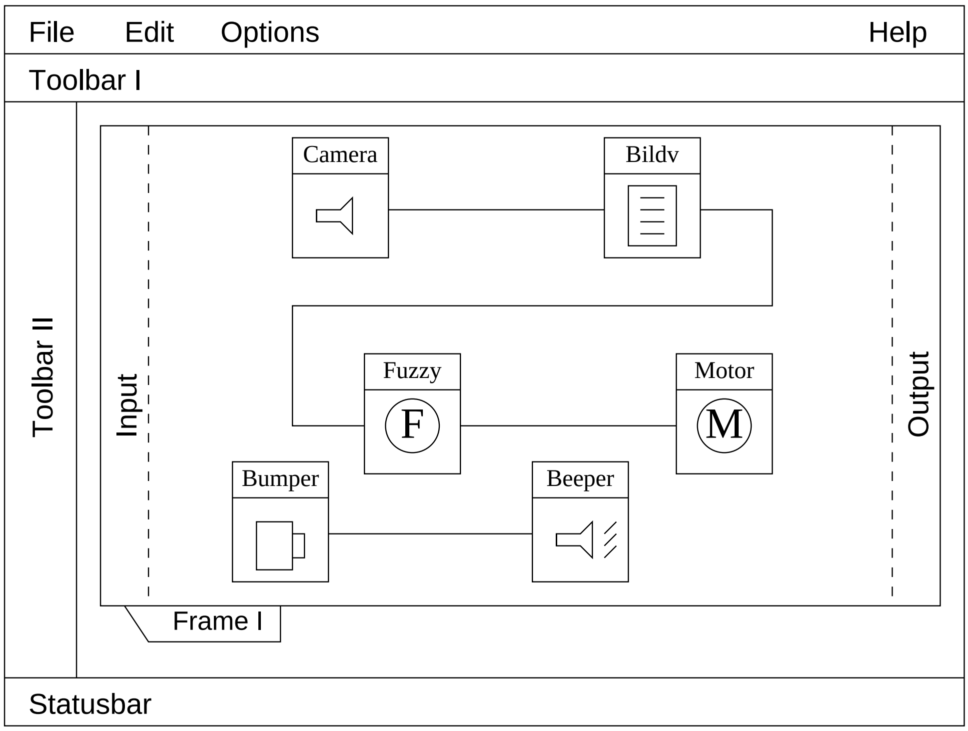

Abbildung 1.3: In diesem Beispielprogramm existieren zwei völlig voneinander unanbhängige Pfade. Der obere Pfad soll ein Bild mithilfe einer Kamera einlesen. Das Bild wird dann in einer Bildverarbeitung umgewandelt und das Ergebnis wird einem Fuzzy-Regler mitgeteilt. Dieser steuert schließlich den Motor des Roboters an. Der zweite Pfad überträgt nur Bumperimpulse an einen Lautsprecher.

Algorithm 1

1. Absolute Priorität haben jene Units die entweder keine Eingänge besitzen, deren Eingangsparameter mit dem Inputfeld des Frames verbunden sind oder deren Eingangsparameter ausnahmslos mit Initialwerten oder Konstanten belegt sind. Sie sind der *Beginn* eines Pfades.

2. Anschließend wird Schrittweise entlang des durch die gerichteten Verbindungen vorgegebenen Pfades entlanggegangen und somit die Numerierung fortgesetzt. Sind mehr als ein Pfad vorhanden, so wird ein beliebiger Pfad ausgewählt, um einen Schritt fortzuschreiten. Danach wird der nächste Pfad ausgewählt und genauso verfahren.

3. Ist das *Ende* eines Pfades erreicht, so wird dieser vorübergehend stillgelegt, bis alle übrigen Pfade abgearbeitet sind. Danach sind alle Units numeriert worden. Das Ende eines Pfades ist erreicht, wenn

   - es keine weiteren Connections mehr gibt
   - das Outputfeld des Frames erreicht worden ist
   - das Innere Gebiet einer Kontrollstruktur (sh. später) betreten wird
   - oder die nachfolgende Unit schon durch einen anderen Pfad erschlossen und numeriert worden ist.

4. Teilt sich ein Pfad in zwei oder mehrere Pfade so gibt es wiederum zwei Strategien:

   (a) Der geteilte Pfad gliedert sich in zwei oder mehrere Subpfade, deren virtuelles Erscheinungsbild wieder einen einzigen Pfad ergibt. Bei der Numerierung kommt dieser virtuelle Pfad nach oben genannter Strategie zum Zug; intern kommt allerdings erst der erste, beim nächsten Numerierungschritt der zweite usw. dran.

   (b) Die durch die Teilung neu entstandenen Pfade werden völlig gleichwertig in die bereits vorhandenen Pfade eingefügt und kommen wie gehabt bei der Numerierung an die Reihung.

5. Bei rekursiven Verbindungen entscheidet über eine weitere Numerierung dieser rekursiven Verbindung die Tatsache ob die an dieser rekursiven Verbindung anliegende Unit schon berücksichtigt worden ist oder nicht. Eine rekursive Verbindung ist also nur ein Spezialfall eines endenden Pfades.

6. Tritt eine Super-Unit auf so sind wieder zwei Strategien möglich:

   (a) Die Unit wird in seine Bestandteile aufgelößt und in das Mainframe virtuell hineinkopiert. Die so neu entstandenen Units werden so wie bereits in der Mainframe vorhandene Units behandelt.

   (b) Die Unit bleibt weiterhin als Einheit geschlossen und wird bei der Abarbeitung vollständig berechnet, sodaß am Ausgang alle Werte anliegen. Intern wird wieder wie beginnend mit Punkt 1) verfahren.

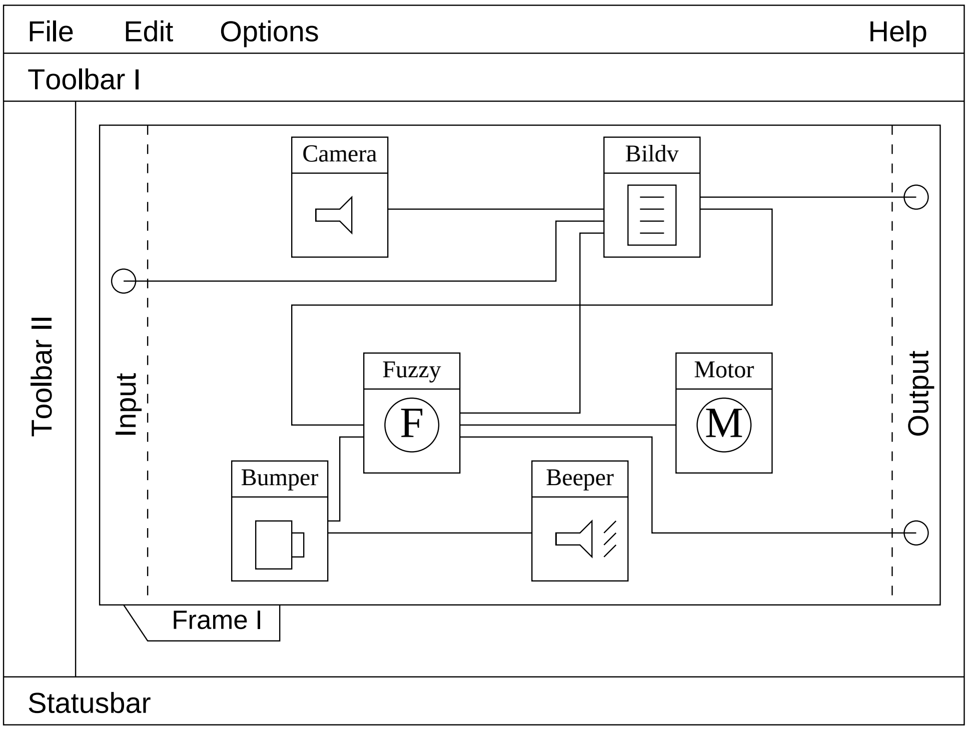

Abbildung 1.4: Erweiterung des Beispiels in der Abb. 1.3. Es soll hier weniger ein praktisches Beispiel als vielmehr die Möglichkeiten komplizierter Verknüpfungen (wie Rekursion, etc.) dargestellt werden.

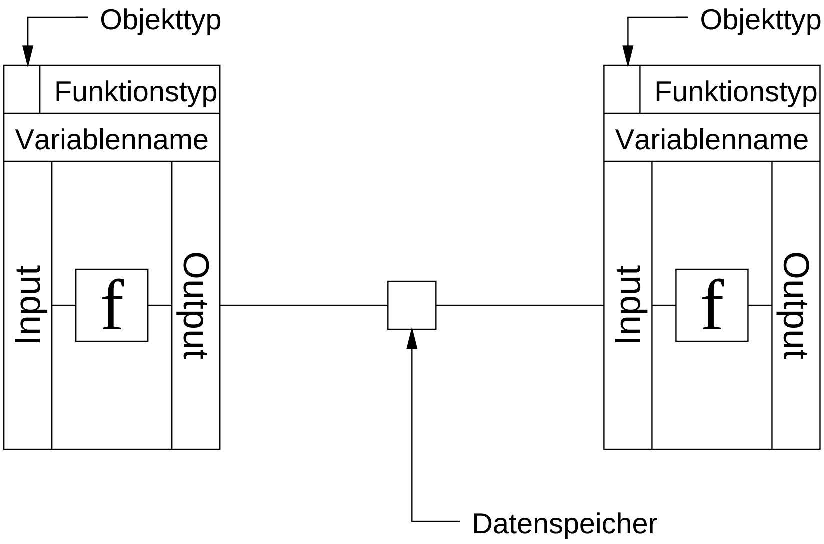

Abbildung 1.5: Schematische Darstellung zu Datenübertragung zwischen zwei Units: Das kleine Viereck soll den einelementigen Datenspeicher darstellen, indem die Daten einer Unit an eine andere weitergegeben werden. Natürlich entstehen bei mehreren Connections mehrere solche Datenspeicher.

Algorithm 2

1. Eine Scheduler zur Verwaltung der einzelnen Unit-Aufrufe wird gestartet. Dieser wählt nun die einzelnen Units zur Berechung ihrer Ausgangsparameter anhand der vergebenen ReihungsID aus. Ist das letzte Element erreicht, so wird danach wieder von vorne begonnen.

2. Bevor eine Berechnung einer gewählten Unit gestartet wird werden zwei Bedingungen überprüft (Abarbeitungs- oder Ausführbedingung):

   (a) An allen Eingängen liegt ein Wert an. Ein Wert liegt an, wenn die Datenspeicher zwischem der vorgeschalteten Units und der ausgewählten Unit belegt ist, ein Initialwert oder eine Konstante angegeben wurde, oder der Unit durch das Inputfeld des Mainframes ein Argument übergeben worden ist. Handelt es sich um eine Unit deren Eingang mit dem Inputfeld des Subframes verbunden ist, so liegt dann ein Wert an, wenn die im übergeordneten Frame eingesetzte Super-Unit die gleichen Bedingungen wie eine normale Unit vorfindet.

   Sind keine Eingänge vorhanden, so ist die Bedingung immer erfüllt.

   Besitzt eine Unit eingehende Steuerleitungen (Connections vom Typ `intern`), so werden diese Eingänge zuerst überprüft und ausgewertet. Steurleitungen können Objekteigenschaften zu Laufzeit aktiv verändern, was u. U. bewirken kann, daß einzelne herkömmliche Eingangsparameter deaktiviert werden. Diese deaktivierten Eingängen sind nun von der oben formulierten Bedingung zur Überprüfung der Eingänge ausgenommen.

   (b) Die zwischen der gewählten Unit und der nachgeschalteten Units befindlichen Datenspeicher ist leer bzw. einer oder mehere der Datenspeicher ist nicht vorhanden (d. h. es besteht keine Connection an einem vorhandenen Output-Pin). Sonderfälle werden wie in (a) behandelt.

   Sind keine Ausgänge vorhanden, so ist die Bedingung immer erfüllt.

3. Sind die beiden Bedingungen von 2. erfüllt, so wird die Berechnung der Ausgänge anhand der Eingänge und der Übertragungsfunktion berechnet. Andernfalls wird bei 2. mit der nächsten vom Scheduler ausgewählten Unit fortgesetzt.

4. Alle Datenspeicher zwischen der vorgeschalteten und der ausgewählten Unit werden gelöscht. Weiters werden die Ergebnisse der Berechungen in die zugehörigen Datenspeicher der Ausgänge geschrieben. Es ist erlaubt, daß nicht alle oder auch keine Werte in die nachfolgenden Datenspeicher geschrieben werden (z.B. Component-Unit Kamera liefert nur alle 200ms ein Bild).

5. Nun wird bei 2. mit einer vom Scheduler neu gewählten Unit fortgesetzt.

1.6 Kontrollstrukturen:

Die Programmiersprache soll auch die sprachüblichen Kontrollstrukturen aufweisen. Dazu gehören:

- if - Kontrollstruktur (mit nicht ausschließenden Nachfolgepfaden)
- if - Kontrollblock (mit ausschließenden Nachfolgepfaden)
- while - Kontrollblock
- do-while - Kontrollblock
- for - Kontrollblock

Bei den Kontrollstrukturen ist also weiters zu unterscheiden:

- Einelementige Kontrollstruktur (hier einfach Kontrollstruktur)
- Mehrelementige Kontrollstruktur (Kontrollblock)

Die einelementige Kontrollstruktur - besteht im wesentlichen aus einer Special-Unit (hier *Kontroll*-Unit). Diese Unit weist im Gegensatz zu herkömmlichen Units etwas flexiblere Abarbeitungsvorschriften sowie eine ganz spezifische interne Übertragungsfunktion auf. Die Eigenschaften dieser Übertragungsfunktion soll im wesentlichen über Optionsfenster der jeweiligen Unit spezifiziert werden können (d.h. kein Java-Programm o. ä.).

Auch ist es bei solchen Unit möglich über *Steuerleitungen* (Connections vom Typ `intern`) gewisse Input-Pins (Eingangsparameter) einer Unit ein- bzw. auszuschalten, was sich vor allem bei der Abarbeitungsbedingung auswirkt, da ausgeschaltete Pins nicht in dies Bedingung miteinfließen. Auch kann eine solche Steuerleitung durch einen spezifischen Wert die Berechung der Übertragungsfunktion unterdrücken. Wichtig ist, daß immer erst der Wert der Steuerleitung eingelesen und ausgewertet wird, bevor die anderen Input-Pins gelesen werden. Wenn weiters die Übertragungsfunktion nicht ausgeführt wird, so wird auch wie in Algorithmus 2 speziefiziert, der Datenspeicher zwischen der aktuellen und der vorgeschalteten Unit nicht gelöscht, d. h. auch Werte der Steuerleitungen liegen weiter an der jeweiligen Unit an.

Es kann weiters vorkommen, daß solche Units auch interne Variablenwerte statisch halten; d. h. daß Werte zwischen zwei erfolgreichen Abarbeitungen der Übertragungsfunktion gespeichert werden.

Die mehrelementige Kontrollstruktur - oder *Kontrollblock* besteht aus mehreren durch Steuerleitungen miteinander verknüpften Kontroll-Units. Sie bilden somit einen *Block*. Ein Block ist graphisch gesehen ein begrenztes Areal in dessen Inneren zusätzliche Gesetze bei der Plazierung von Units und Verbinden von Units durch Connections herrschen können. Tritt eine beliebige Connection direkt in einen solchen Block ein, so bedeutet dies das Ende eines Pfades.

Optisch bilden im folgenden die Steuerleitungen diese optischen Begrenzungen, dies ist aber willkürlich und kann in der entgültigen Benutzer-IDE anders gelöst werden. Die Steuerleitungen selbst existieren immer und es kann immer nur der gesamte Block, nie aber eine einzelne Units daraus plaziert werden.

1.6.1 Die if - Kontrollstruktur

Diese einelementige Kontrollstruktur könnte prinzipiell auch als normale Unit realisiert sein. Um eine gewisse konzeptuelle Einheit zu waren, wird sie jedoch zu den Kontrollstrukturen hinzugerechnet.

Bei der if-Kontrollstruktur handelt es sich um die primitivste denkbare bedingte Verzweigung in der GPS. Sie besitzt keine Ein- oder Ausgänge für Steuerleitungen.

Die Unit kann beliebig viele normale Eingänge besitzen. In dem Optionenfenster der Unit können diese Inputs durch ihere Bezeichner logisch miteinander verknüpft werden. Bei der logischen Verknüpfung werden die standartmäßigen Verknüpfungsoperatorern wie `AND, OR, EXOR, NOT` etc. zur Verfügung gestellt.

Es können in diesem Optionenfenster beliebig viele Bedingungen formuliert werden. Aus der Anzahl der Bedingungen ergibt sich dann die Anzahl der Output-Pins plus einem zusätzlichen else-Pin der Kontroll-Unit. Standartmäßig ist der Typ der Ausgangsparameter `boolean`. Dies kann jedoch vom Programmierer für jeden Pin einzeln verändert werden.

Ist nun bei der Abarbeitung die if-Kontroll-Unit an der Reihe, so wird wie in Algorithmus 2 definiert die Abarbeitungsbedingung überprüft. Ist diese erfüllt, so werden anhand der Eingangswerte die formulierten logischen Werte der bereits definierten Bedinungen berechnet. Ist für die Ausgansparameter kein bestimmter Wert und Typ spezifiziert, so wird im Falle des Werts `true` einer bestimmten Bedingung, der korrespondierende Ausgangspin mit dem Wert `true` beschrieben. Ist keine der Bedingungen „wahr", so wird der else-Pin mit einem Wert beschrieben.

Dieser geschriebene Wert kann aber vom Programmierer noch beliebig näher (durch eine Konstante oder einen arithmetischen Ausdruck) für jede Bedingung definiert werden.

Wesentlich bei der einfachen if-Kontrollstruktur ist, daß alle Bedingungen auf ihren Wahrheitswert überprüft werden und somit beliebig viele nachgeschaltete Pfade parallel weiterverfolgt werden können. Es handelt sich also um eine nicht ausschließende oder-Verknüpfung der Bedingungen.

1.6.2 Der if - Kontrollblock

Der *if-Kontrollblock* (siehe Abb. 1.6 und Abb. 1.7) besteht nun aus zwei durch Steuerleitungen verbundene Kontroll-Units:

- Die if-Kontrollunit und
- die end-if-Kontrollunit.

Die if-Kontroll-Unit: Diese Unit stimmt im wesentlichen mir der in 1.6.1 definierten Kontroll-Unit überein. Es bestehen aber auch einige gravierenden Unterschiede:

1. Die Unit besitzt eine eingehende und eine ausgehende Steuerleitung. Der interne Typ der eingehenden Connection ist wie ein Verbindung vom Typ `boolean` realisiert; die ausgehende Steuerleitung ist intern vom Typ `integer`. Liegt ein Wert an der eingehenden Steuerleitung an, so wird wie in 1.6.1 verfahren, sonst ist klarerweise die Ausführbedinung nicht erfüllt.

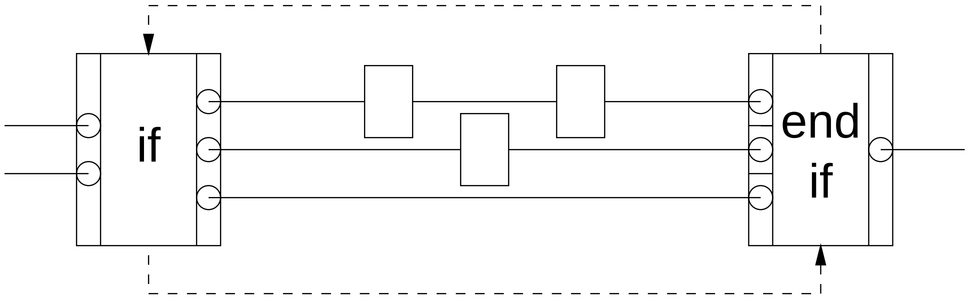

Abbildung 1.6: Einfacher Anwendungsfall eines if - Kontrollblockes.

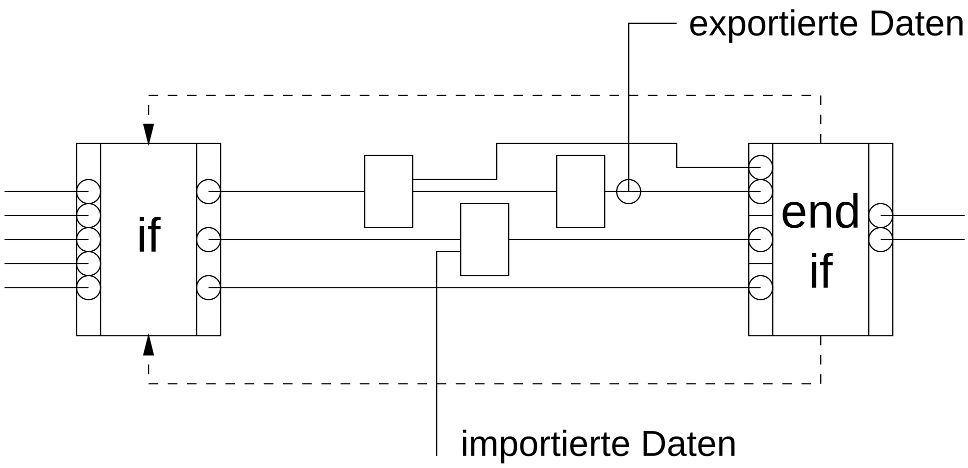

Abbildung 1.7: Komplizierterer, durch angegebene Definition des if-Kontrollblocks mit maximaler Funktion versehener Anwendungsfall.

2. Bei der Berechung des Wahrheitswertes der im Optionenfenster der Unit formulierten Bedingungen wird solange beginnend mit der ersten Bedingung solange fortgefahren, bis eine wahre Bedingung (falls keine „wahr“ ist so wird wieder der else-Pin mit einem Wert beschrieben) erreicht ist. Nun wird abgebrochen und nur der zu dieser Bedingung korrespondierende Ausgang gesetzt. Die restlichen Ausgänge bleiben unberücksichtigt. Die Bedingungen sind also im Gegensatz zu 1.6.1 durch ein ausschließendes „oder“ miteinander verknüpft.

3. Nach der Überprüfung der Bedingungen wird an der ausgehenden Steuerleitung ein mit der wahren x-ten Bedingung korrespondierender Integer-Wert x geschrieben. Ein Ausnahme hiervon wäre dann gegeben, wenn am x-ten Ausgang der Unit sich keine Connection befindet. Dann wird kein Wert auf die ausgehende Steuerleitung geschrieben.

Die end-if-Kontroll-Unit: Diese Unit ist etwas komplizierter aufgebaut, um möglichst flexible Programmiermöglichkeiten zu erreichen. Sie besitzt beliebig, wie immer von dem Programmierer konfigurierbare Eingangsparameter. Diese Parameter können weiters in Gruppen zusammengefaßt werden. Jeder dieser Gruppe wird ein Integer-Wert zugeordnet. Optisch sind diese Gruppen in Abb.1.6 und Abb. 1.7 durch Strichunterteilungen getrennt.

Es kann immer nur eine Gruppe zur Bewertung der Ausführbedingung herangezogen werden. Welche Gruppe ausgewählt wird, wird durch die eingehende Steuerleitung entschieden.

Die Anzahl und Typ der Ausgänge sind ebenfalls variabel und können vom Programmierer eingestellt werden. Im Optionenfenster kann dann (eventuell graphisch) festgelegt werden, welcher Eingang mit welchem Ausgang korrelliert. Hier soll auch noch eine Standartvorgebung realisiert werden, um dem unbedarften Benutzer Arbeit abzunehmen. Diese Vorgebung ergibt sich aus folgenden Punkten:

- Die Anzahl der Ausgänge ergibt sich wie folgt: Es gibt die Menge T alle an den Eingängen vorkommenden Typen t. Die Anzahl der verschiedenen Typen über alle Eingänge ist $n = \mid T \mid$ und es gibt eine Funktion g, die jedem Element aus T eine eindeutige natürliche Zahl m zuordnet, wobei $0 < m \leq n$ gilt. Nun gibt es eine Funktion f die jedem $k \in \{y \mid 0 < y \leq n\}$ einen Wert q zuordnet, die maximale Anzahl eines bestimmten Typs innerhalb einer Gruppe über alle Gruppen angibt. Die Summe $\sum_{k \in \{y \mid 0 < y \leq n\}} f(k)$ ergibt dann die Anzahl der Ausgänge. Der Typ der einzelnen Ausgangsparameter ergibt sich, indem jedem an dem Eingang vorkommenden Typ t eine Anzahl von k Ausgängen des zugehörigen Typs zugeordnet wird, wobei gilt $k = f(g(t))$.

- Es wird jeder Eingangspin der Unit intern mit einem Ausgangspin gleichen Typs verbunden, wobei Eingangsparameter derselben Eingangsgruppe alle verschiedene Ausgangspins aufweisen, während die Verbindungen zu den Ausgängen über alle Gruppen nicht disjunkt sein müssen.

Ist die Ausführbedingung erfüllt so werden zuerst die korrellierenden Ausgangspins mit den Werten der ausgewählten Eingangsgruppe beschrieben. Anschließend wird auf der ausgehenden Steuerleitung (die sich wie eine Connection vom Typ `boolean`) verhält der Wert `true` geschrieben.

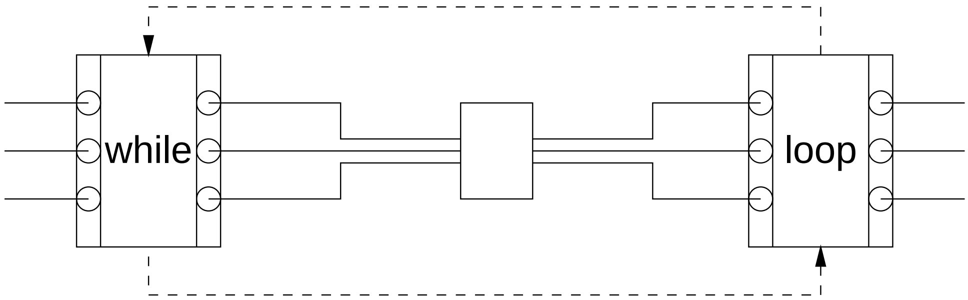

Abbildung 1.8: Einfacher Anwendungsfall des while-Unit

Allgemeines: Im Inneren des Strukturblocks, dem Strukturgebiet, ergeben sich nun von der Anzahl x der Ausgänge der if-Unit x neue Pfade. Da immer nur ein Pfad gleichzeitig abgearbeitet wird, ist es nicht sinnvoll, die Pfade untereinander mit Connections zu verbinden. Das gilt natürlich nicht für die neuen Pfade, die entstehen, wenn einer der vorher genannten Pfade sich erst später wieder in neue Pfade spaltet.

Auch müssen alle Endigungen dieser unmittelbar nach der if-Unit entstehenden Pfade in einer Gruppe der end-if-Unit enden, um eine Blockierung bei der Ausführung auszuschließen. Es ist zu diskutieren, inwieweit man dem Programmierer daran hindern soll, solche Programmzustände zu formulieren.

Die Steuerleitung von der end-if-Unit zur if-Unit hat einen Initialwert, der intern den Wert `true` besitzt.

Es ist weiters erlaubt, Daten außerhalb des Kontrollgebietes zu importieren oder zu exportieren (sh. Abb. 1.7).

Wie sich nun der gesamte if-Kontrollblock zur Ausführungszeit verhält ist in Algorithmus3 beschrieben.

1.6.3 Der while - Kontrollblock

Der *while - Kontrollblock* besteht wie der if - Kontrollblock aus zwei durch Steuerleitungen verbundene Special-Units (Kontroll-Units):

- Der while-Kontroll-Unit und
- der loop-Kontroll-Unit.

Die while-Kontroll-Unit: - besitzt beliebig viele Eingangsparameter, die der Programmierer ständig erweitern und reduzieren kann. Die Parameter können aus beliebigen Typen bestehen.

In einem Optionenfenster der while-unit können diese Eingänge logisch zu einer Bedingung (der Schleifenbedinung) verknüpft werden. Hierbei sollen die Verknüpfungregeln genauso wie bei der if-Unit realisiert sein, mit der Ausnahme, daß nur eine Bedingung definiert werden kann.

Weiters ist die Anzahl und Typ der Outputpins der while-Unit identisch mit den der Inputpins.

Zudem besitzt die while-Unit eine eingehende und eine ausgehende Steuerleitung vom Typ `intern`. Diese besteht intern eigentlich aus zwei Steuerleitungen,

Algorithm 3

1. Falls die if-Kontrollunit aus dem if-Kontrollblock vom Scheduler ausgewählt wird, so wird die Ausführbedingung überprüft. Da der Wert der eingehenden Steuerleitung einen Initialwert aufweist, ist bei der ersten Anwählung dieser Unit die Ausführbedingung nach den herkömmlichen Regeln erfüllt.

   Ist die Ausführdingung erfüllt und für keine andere Unit im Kontrollblock ist die Ausführbedingung jemals erfüllt gewesen, so wird der if-Kontrollblock zum ersten mal ausgeführt. Die Ausführbedingung kann weiters nach der Definition des if-Kontrollblockes logisch nur dann erfüllt sein, wenn keinerlei Daten zwischen Units innerhalb des Blocks gespeichert sind (ausgenommen sind natürlich die importierten Daten).

   Ist die Bedingung nicht erfüllt, so ist die Abarbeitung des if-Kontrollblocks beendet.

2. Die in der if-Unit formulierten Bedingungen werden überprüft und die entsprechenden Ausgänge der if-Unit beschrieben. Weiters wird auch die ausgehende Steuerleitung mit einem Wert beschrieben. Die if-Unit kann so lange nicht mehr ausgeführt werden, bis die eingehende Steuerleitung beschrieben wird.

3. Der ausgewählte Pfad innerhalb des Blocks wird abgearbeitet. Die einzelnen Units werden laut Algorithmus 1 und 2 ohne gesonderte Unterscheidung, ob es sich um eine Unit innerhalb oder außerhalb eines Blocks handelt, ausgewählt und abgearbeitet. Falls der ausgewählte Pfad nicht existiert (es sind keine Units angeschlossen), so wird, da kein Wert an die ausgehende Steuerleitung des if-Unit geschrieben wird, die Abarbeitung des if-Kontrollblocks beendet.

4. Ist das Ende des Pfades erreicht, wird die end-if-Unit abgearbeitet, da an der eingehenden Steuerleitung bereits ein Wert anliegt und dieser die zu untersuchenden Eingänge auswählt.

5. Die Ausgänge der end-if-Unit werden beschrieben. Auch wird ein Wert auf die ausgehende Steuerleitung geschrieben. Die Abarbeitung des if-Kontrollblocks ist beendet und da nun wieder ein Wert an der eingehenden Steuerleitung der if-Unit anliegt kann einer erneuter Start zu Abarbeitung des if-Kontrollblocks beginnen.

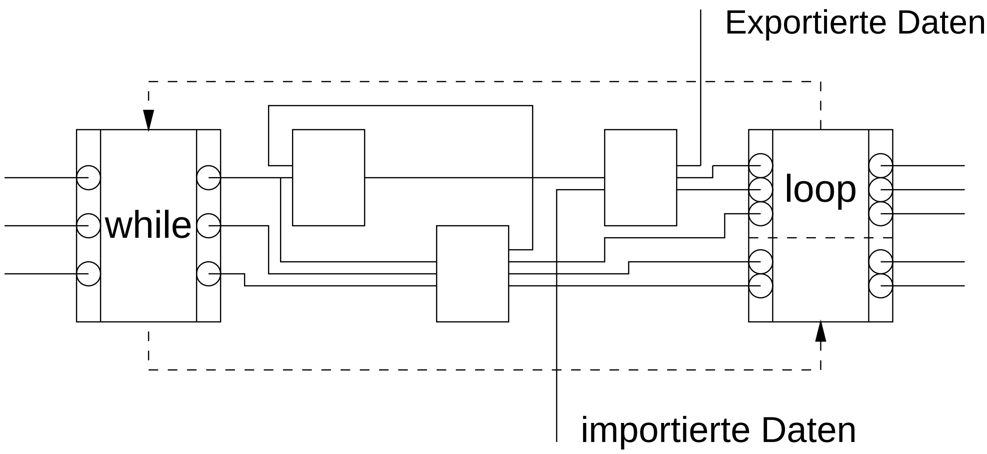

Abbildung 1.9: Komplizierterer, durch angegebene Definition des while-Kontrollblocks mit maximaler Funktion versehener Anwendungsfall.

die zum einen aus einem Boolschen und zum anderen aus einer Struktur besteht, dessen Zusammensetzung identisch mit der der Eingänge der while-Unit ist. Der Boolsche Wert besitzt den Initialwert `false`.

Die ausgehende Steuerleitung ist vom Typ her genauso, wie die eingehende Steuerleitung definiert.

Liegt nun an allen Eingängen ein Wert an UND ist der Boolsche Wert der eingehenden Steuerleitung mit einem Wert `false` belegt, so werden diese Werte in einem *internen* Datenspeicher der while-Unit geschrieben.

Ist dieser Boolsche Wert `true`, so wird, falls Werte in der Struktur der Steuerleitung anliegt, die Werte der Struktur auf den internen Datenspeicher geschrieben.

Dieser Boolsche Wert fungiert also zum steuern eines Demultiplexers. Man könnte zudem diese interne Steuerleitung als zwei normale Connections realisieren, mit der Ausnahme, daß die Boolsche Connections bei der standartmäßigen Überprüfung der Ausführbedingung bei Units (wie in Algorithmus 2 beschrieben) eine Sonderbehandlung erfährt. Die Übergabe der Werte dieser Sonderleitungen funktioniert aber wie im oben genannten Konzept der Parameterübergabe zwischen zwei Units. Da nun dieser Sondereingang speziell behandelt wird, ist die Ausführbedingung dieser Unit nie erfüllt falls kein Boolscher Wert anliegt und es werden keine Werte, weder von den normalen Eingängen, noch von der Struktur der Sonderleitungen eingelesen.

Ist andererseits der Boolsche Wert zwar belegt, die Eingänge sind oder die Struktur der Steuerleitung ist jedoch nicht belegt, so ist die Ausführbedingung ebenfalls nicht erfüllt. Da aber die einelementigen Zwischenspeicher der Connections erst dann gelöscht wird, wenn die nachgeschaltete Unit ausgeführt wurde, so bleibt aber der Boolsche Wert der Steuerleitung weiterhin anliegend.

Ist die Ausführbedingung nun erfüllt, wird die in der while-Unit formulierte Schleifenbedingung anhand der zwischengespeicherten internen Werte auf ihren Wahrheitsgehalt überprüft. Der Boolsche Teil der ausgehenden Steuerleitung wird daraufhin mit dem Ergebnis dieser Berechnungen belegt.

Ist der Wert `true`, so werden die Ausgänge der while-Unit mit den Werten des internen Speichers belegt. Ist der Wert jedoch `false`, so wird am Ausgang nichts geschrieben, die Struktur der ausgehenden Steuerleitung aber mit den internen Speicherwerten belegt.

Besitzt keine der Ausgänge der while-Unit Connections zu weiteren Units, so wird nichts in die ausgehende Steuerleitung der Unit geschrieben.

Die loop-Kontroll-Unit: Die Anzahl und der Typ der Eingänge der loop-Unit ist im einfachsten Fall mit Anzahl und Typ der Eingänge in der while-Unit identisch und auch davon direkt abhängig. Das heißt, daß eine Veränderung der Eingänge der while-Unit eine Veränderung der Eingänge der loop-Unit bewirkt. Die Umkehrung soll aufgrund der Übersichtlichkeit beim Programmieren selbst, als auch wegen Design-Vorteilen beim Entwurf dieser GPS nicht möglich sein. Zusätzlich kann der Programmierer noch unabhängige Eingänge hinzufügen. Diese unabhängigen Eingänge sollen auch optisch von den anderen Eingängen getrennt sein (in Abb. 1.8 und Abb. 1.9 durch den Strich im Inputfeld angedeutet), auf die der Programmierer keinerlei modifizierenden Einfluß haben soll.

Anzahl und Typ der Ausgänge setzen sich weiters, ähnlich wie bei der while-Unit aus den Eingängen der loop-Unit zusammen.

Zudem besitzt die loop-Unit wiederum eine eingehende und eine ausgehende Steuerleitung. Der Typ ist mit den der Steuerleitungen der while-Unit identisch. Die Entitäten der eingehenden Steuerleitung besitzen keinen Initialisierungswert.

Liegt nun am Boolschen Teil der eingehenden Steuerleitung der Wert `true` an, so werden alle fixen Eingänge (also alle jene Eingänge deren Existenz von der while-Unit abhängen) auf das Array der ausgehenden Steuerleitung geschrieben. Weiters wird der Boolsche Teil auf `true` gesetzt. Die eventuell vorhandenen Werte der zusätzlichen Eingänge werden eingelesen und statisch innerhalb der loop-Unit zwischengespeichert.

Liegt am Boolschen Teil der eingehenden Steuerleitung jedoch der Wert `false` an, so wird das Array der eingehenden Steuerleitung auf die korrespondierenden Ausgänge geschrieben. Weiters werden die eventuell zwischengespeicherten Werte der zusätzlichen Eingänge der loop-Unit auf die restlichen Ausgänge übertragen. Abschließend wird der Boolsche Wert der ausgehenden Steuerleitung mit `false` belegt.

Allgemeines: Prinzipiell gibt es im Gegensatz zum if-Kontrollblocks im Inneren der Kontrollblocks keine verschaltungstechnischen Einschränkungen. Weiters hängen alle oben genannten Aktionen natürlich von der Erfüllung der Ausführbedingung nach Algorithmus 2 ab.

Wie nun der gesamte Block zur Ausführungszeit abgearbeitet wird, ist in Algorithmus 4 beschrieben

1.6.4 Der do-while - Kontrollblock

Naturgemäß ist der *do-while-Kontrollblock* ähnlich dem while-Kontrollblock aufgebaut (siehe 1.6.3). Dieser Block (siehe Abb. 1.10 besteht aus einer

- do-Kontroll-Unit und einer

Algorithm 4

1. Falls die while-Kontrollunit aus dem while-Kontrollblock vom Scheduler ausgewählt wird, so wird die Ausführbedingung überprüft. Da der Boolsche Teil der eingehenden Steuerleitung den Initialwert `false` aufweist, ist bei der ersten Anwählung dieser Unit die Ausführbedingung nach den herkömmlichen Regeln erfüllt.

   Ist die Ausführbedingung erfüllt und für keine andere Unit im Kontrollblock ist die Ausführbedingung jemals erfüllt gewesen, so wird der while-Kontrollblock zum ersten mal ausgeführt. Die Ausführbedingung kann weiters nach der Definition des while-Kontrollblockes logisch nur dann erfüllt sein, wenn keinerlei Daten zwischen Units innerhalb des Blocks gespeichert sind (ausgenommen sind natürlich die importierten Daten).

   Die Ausführbedinung ist weiters erfüllt sein, wenn der Boolsche Teil der eingehenden Steuerleitung den Wert `true` besitzt.

   Ist die Ausführbedingung nicht erfüllt, so ist die Abarbeitung des while-Kontrollblocks beendet.

2. Ist die Ausführbedingung erfüllt so werden die entsprechenden Daten (entweder von den Eingängen oder von der Steuerleitung) in den internen Speicher der while-Unit geschrieben.

3. Die in der while-Unit formulierte Bedingung wird überprüft.

   (a) Falls die Bedingung „wahr“ ist, so werden Ausgänge beschrieben. Weiters wird auch der Boolsche Teil der ausgehenden Steuerleitung mit `true` beschrieben. Die if-Unit kann so lange nicht mehr ausgeführt werden, bis die eingehende Steuerleitung beschrieben wird. Es wird mit 4. fortgesetzt.

   (b) Falls die Bedingung „falsch“ ist, so wird der Boolsche Teil der ausgehenden Steuerleitung mit `false` beschrieben und die statisch gespeicherte Struktur des while-Unit auf den anderen Teil dieser Steuerleitung übertragen. Es wird mit 6. fortgesetzt.

4. Die Pfade innerhalb des Blocks werden abgearbeitet. Die einzelnen Units werden laut Algorithmus 1 und 2 ohne gesonderte Unterscheidung, ob es sich um eine Unit innerhalb oder außerhalb eines Blocks handelt, ausgewählt und abgearbeitet.

5. Ist das Ende des Pfades erreicht, wird die loop-Unit abgearbeitet, da am Boolschen Teil der eingehenden Steuerleitung bereits ein Wert anliegt.

6. Ist der Wert des Boolschen Teils der eingehenden Steuerleitung der loop-Unit wird untersucht:

   (a) Ist dieser Wert `true`, so wird der Wert der Struktur der eingehenden Steuerleitung an die Struktur der ausgehenden Steuerleitung übertragen. An dem boolschen Teil der ausgehenden Steuerleitung wird der Wert `true` angelegt, sodaß bei der nächsten Abarbeitung der while-Unit nicht die Daten an den Eingängen der Unit verwendet werden.

   Anschließend wird wieder mit 1. fortgesetzt.

   (b) Ist dieser Wert `false`, so wird der Wert der eingehenden Steuerleitung, sowie die der in der loop-Unit zwischengespeicherten Werte der zusätzlichen Eingänge derselben, auf die Ausgänge übertragen. Der Wert des Boolschen Teils der ausgehenden Steuerleitung erhält den Wert `false`, um die Eingangsparameter der while-Unit wieder zu aktivieren.

   Die Abarbeitung des while-Kontrollblocks ist beendet.

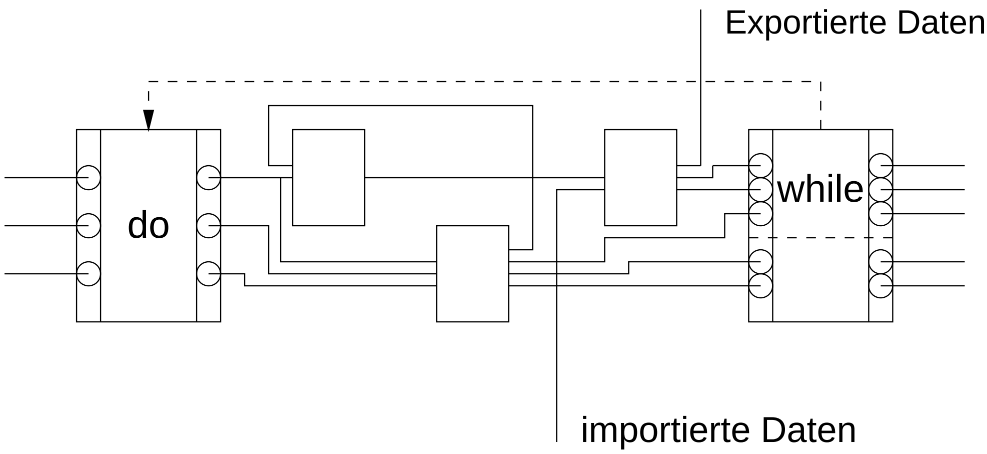

Abbildung 1.10: Kompliziertes Anwendungsbeispiel, um die volle Funktionalität des do-while-Kontrollblocks darzustellen.

- while-Kontroll-Unit.

Da das Prinzip der Verschaltung an den vorigen Beispielen genug erleutert wurde, wurde hier auf das einfache Anwendungsbeispiel verzichtet.

Die do-Kontroll-Unit: Die Eingangs- und Ausgangsparameter sind wie in der while-Unit im Abschnitt 1.6.3 definiert. Weiters besitzt die Kontroll-Unit nur eine eingehende, aber keine ausgehende Steuerleitung. Die eingehende Steuerleitung ist ebenfalls wie in der while-Unit im Abschnitt 1.6.3 definiert. Der Boolsche Teil der eingehenden Steuerleitung weist wiederum den Initialwert `false` auf.

Auch spielt dieser Boolsche Teil wieder die Rolle eines Demultiplexers: Ist der Wert `false`, so werden die normalen Eingangsparameter überprüft; ist der Wert `true`, so wird die Struktur in der Steuerleitung eingelesen. Ist die Ausführbedingung erfüllt, so besteht die Übertragungsfunktion nur darin, entweder die Werte der normalen Eingänge oder die Werte der Struktur auf die Ausgänge zu übertragen.

Die while-Kontroll-Unit: Auch hier sind die Eingangs- und die Ausgangsparameter wieder wie in der loop-Unit im Abschnitt 1.6.3 definiert. Sie besitzt nur eine ausgehende Steuerleitung vom gleichen Typ wie die do-Unit als eingehende Steuerleitung aufweist. Zusätzlich kann in einem Optionenfenster wie in der while-Unit des Abschnitts 1.6.3 eine Bedingung formuliert werden. Da die Eingänge wieder in zwei optisch getrennten Gruppen aufgeteilt wird, können für die Formulierung der Bedingung nur die erstere (hier wieder von der do-Unit abhängige) Gruppe herangezogen werden. Die Eingänge der zweiten Gruppe können vom Programmierer wieder beliebig hinzugefügt werden.

Ist die Ausführbedingung erfüllt (hier von keiner Steuerleitung abhängig), so wird die formulierte interne Bedingung anhand der Eingangswerte auf ihern Wahrheitswert überprüft. Hat dieser den Wert `true`, so werden die Eingangsdaten der abhängigen Gruppe (erste Gruppe) auf die Struktur der ausgehenden

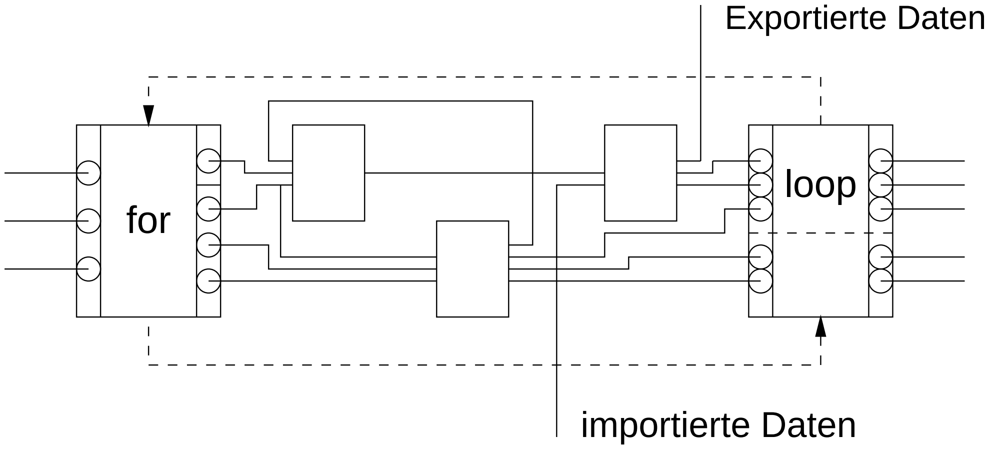

Abbildung 1.11: Kompliziertes Anwendungsbeispiel, um die volle Funktionalität des for-Kontrollblocks darzustellen.

Steuerleitung übertragen. Weiters ist der Wert der Boolschen Teils `true`. Es ist zu beachten, daß die Werte der zweiten Gruppe verworfen werden.

Ist der Wert der formulierten Bedingung `false`, so werden die Ausgangsparameter mit den Werten der Eingangsparameter belegt. Der boolsche Teil der Steuerleitung erhält den Wert `false`.

Allgemeines: Da der do-while-Kontrollblock nur eine Steuerleitung besitzt, ist die Abgrenzung des inneren Gebietes in der Abb. 1.10 nicht so deutlich zu erkennen. Da dies aber sowieso nie ein Kriterium für eine Abgrenzung war, kann dies im fertiggestellten grafischen User-Interface durch andere optische Maßnahmen ersetzt werden.

Wie der do-while-Kontrollblock zur Laufzeit abgearbeitet wird, ist in Algorithmus 5 festgelegt.

1.6.5 Der for - Kontrollblock

Im Gegensatz zu der Programmiersprache C soll die for-Schleife (bzw. der for-Kontrollblock) nur in ihrer einfachsten Variante realisiert werden; d.h. es wird eine Schleifenvariable entweder inkrementiert oder dekrementiert, wobei die Schrittweite variabel sein kann.

Für kompliziertere for-Konstruktionen kann immer noch auf den while-Kontrollblock bzw. den do-while-Kontrollblock zurückgegriffen werden, mit denen sich eigentlich (insbesondere durch die Zulassung von exportierten Daten) alle Sonderfälle von for-Schleifen realisieren lassen sollten.

Der for-Kontrollblock besteht nun wiederum aus zwei Special-Units (Kontroll-Units; siehe Abb. 1.11):

- Die for-Kontroll-Unit und
- die loop-Kontroll-Unit.

Algorithm 5

1. Falls die do-Kontrollunit aus dem do-while-Kontrollblock vom Scheduler ausgewählt wird, so wird die Ausführbedingung überprüft. Da der Boolsche Teil der eingehenden Steuerleitung den Initialwert `false` aufweist, ist bei der ersten Anwählung dieser Unit die Ausführbedingung nach den herkömmlichen Regeln erfüllt.

   Ist die Ausführdingung erfüllt und für keine andere Unit im Kontrollblock ist die Ausführbedingung jemals erfüllt gewesen, so wird der do-while-Kontrollblock zum ersten mal ausgeführt. Die Ausführbedingung kann weiters nach der Definition des do-while-Kontrollblockes logisch nur dann erfüllt sein, wenn keinerlei Daten zwischen Units innerhalb des Blocks gespeichert sind (ausgenommen sind natürlich die importierten Daten).

   Die Ausführbedinung ist weiters erfüllt sein, wenn der Boolsche Teil der eingehenden Steuerleitung den Wert `true` besitzt.

   Ist die Ausführbedingung nicht erfüllt, so ist die Abarbeitung des while-Kontrollblocks beendet.

2. Ist die Ausführbedingung erfüllt so werden die entsprechenden Daten (entweder von den Eingängen oder von der Steuerleitung) auf die Ausgänge der do-Unit geschrieben.

3. Die Pfade innerhalb des Blocks werden abgearbeitet. Die einzelnen Units werden laut Algorithmus 1 und 2 ohne gesonderte Unterscheidung, ob es sich um eine Unit innerhalb oder außerhalb eines Blocks handelt, ausgewählt und abgearbeitet.

4. Ist das Ende des Pfades erreicht, wird die while-Unit abgearbeitet.

5. Die in der while-Unit formulierte Bedingung wird überprüft.

   (a) Falls die Bedingung „wahr" ist, so wird die Struktur der ausgehenden Steuerleitung mit den entsprechenden Eingangsparameter belegt und der Boolsche Teil der ausgehenden Steuerleitung mit `true` beschrieben. Anschließend wird wieder mit 1. fortgesetzt.

   (b) Falls die Bedingung „falsch" ist, so wird der Boolsche Teil der ausgehenden Steuerleitung mit `false` beschrieben und die Eingänge der while-Unit an die Ausgangsparameter übertragen. Die Abarbeitung des do-while-Kontrollblocks wird beendet.

Da der for-Kontrollblock nur ein Spezialfall eines while-Kontrollblocks ist, wird hier auf die nähere Erleuterung der einzelnen Kontroll-Units verzichtet. Die Eigenschaften und Funktionen der einzelnen Units können mit drei Ausnahme exakt vom Abschnitt 1.6.3 übernommen werden, wobei die while-Unit das entsprechende Äquivalent zu der for-Unit ist:

- Eine Ausnahme besteht in der Einführung einer zusätzlichen *Schleifenvariable*. Die Schleifenvariable muß von einem ganzahliger numerischer Typ sein. Wie groß der Wertebereich dieser Variablen ist, soll in einem zu der for-Unit zugehörigen Optionenfenster festgelegt werden können.
- Die Wert Schleifenvariable wird an einem zusätzlichen Ausgang der for-Unit ausgegeben. Dieser Output-Pin soll wiederum optisch von den anderen Ausgängen getrennt sein.
- Weiters erfolgt eine Spezialisierung der im Inneren der for-Unit formulierten Schleifenbedingung. Diese kann nun in einem Optionenfenster etwas anders als in der while-Unit definiert werden. Diese Definition besteht wie bei jeder for-Schleife aus drei Teilen:
  - Einem *Inititialisierungssteil*: Hier wird der Schleifenvariable ein Initialwert zugewiesen. Bei dem Initialwert kann es sich um einen konstanten Wert oder um einen Bezeichner einer der Eingangsparameter handeln (soferne der Typ dieses Parameter konform ist).
  - Der eigentlichen *Schleifenbedingung*: Hier wird der Bereich angegeben, in der der numerische Wert der Schleifenvariablen die Schleifenbedingung erfüllt
  - Der *Inkrementierungsteil*: Hier wird die Schrittweite angegeben, um wieviel die Schleifenvariable bei jedem Durchlauf erhöht wird. Es sind auch negative Werte erlaubt (Dekrementierung). Es ist wesentlich, daß bei der Berechnung der Übertragungsfunktion diese Inkrementierung nicht erfolgt. D.h. wenn der Boolsche Teil der eingehenden Steuerleitung den Wert `false` besitzt und die Ausführbedingung erfüllt ist, so findet eine Unterdrückung der Inkrementierung statt.

Da der aus der obigen Definition ergebende Algorithmus sehr dem im Abschnitt 1.6.3 verwendeten Algorithmus 4 ähnelt und sich die geringfügigen Abänderungen leicht aus der angegebenen Unterschieden zum while-Kontrollblock ergeben, wird hier auf die Formulierung eines neuen Algorithmus' verzichtet.

1.7 Genauere Spezifikation der Unit-Objekttypen

Hier werden die Eigenschaften folgender Unit-Typen genauer festgelegt:

- Component-Units
- Java-Units
- Fuzzy-Units
- Net-Units

1.7.1 Component-Units

In einem Projekt kann jeweils ein *Robotertyp* inkludiert werden. Bei einem Robotertyp handelt es sich um eine Abbildung von Funktionskomponenten wie Sensoren (z.B. Kamera) und Aktoren (z.B. Motor). Ein Robotertyp kann beim Beginn eines Projektes aus einer Datenbank ausgewählt werden. Die Übertragung der Roboterdaten in die Datenbank erfolgt über eine Schnittstelle von dem Robotersimulator DIMAS im XML Format (siehe).

Es werden in dieser Datenbank sämtliche Unit-Attribute wie Anzahl der Eingangs- und Ausgangsparameter, Funktionstyp, sämtliche Bezeichner etc. festgelegt. Diese Attribute können vom Programmierer nicht mehr beeinflußt werden. Die einzige Möglichkeit einer Konfiguration der Component-Units besteht durch das Optionenfenster. Die veränderbaren Optionen sind wiederum in der Datenbank bzw. in den jeweiligen XML-Files definiert. Es ist zu Überlegen welche Eigenschaften der Roboter-Komponenten über das Optionenfenster und welche über zusätzliche Eingangsparameter während des Programmablaufes dynamisch eingestellt werden können (z.B. Helligkeitsregelung der Kamera).

Die dafür notwendigen Konventionen zwischen RoboLang und DIMAS sind erst zu definieren.

Bei der Ausführung ist die Übertragungsfunktion nicht in Form eines herkömmlich generierten Java-Programmes zu verstehen, sondern der Aufruf diese Funktion, soll über eine Schnittstelle zu DIMAS bzw. zu dem physischen Roboter selbst (siehe), die Daten der entsprechenden Roboterkomponente einlesen.

1.7.2 Java-Units

In diesen Units kann nun Java-Code als Definition für die Übertragungsfunktion formuliert werden. Bei der Neuprogrammierung einer Java-Unit wird ein Special-Frame zur Verfügung gestellt. Dieses Frame hat im Fall einer Java-Unit die Eigenschaften eines normalen Editors. Das Optionenfenster umfaßt nur die Standarteinträge, wie Anzahl und Bezeichner der Eingangs- und Ausgangsparameter sowie den Bezeichner für Funktionstyp.

Beim Editor wird weiters zwischen einem editierbaren und einem nicht editierbaren Bereich unterschieden. Diese Bereiche sollen optisch (z.B. durch besonders kennzeichnende Hintergrundfarben) getrennt sein.

Der Grund für diese Trennung ist, daß die Eingangs- und Ausgangsparameter im Java-Programm selbst sozusagen als globale Variablen fix definiert sind. Der Scheduler weist den Eingangsvariablen vor der Ausführung, falls die Abarbeitungsbedingung erfüllt ist, Initialisierungswerte aus dem der Unit vorgeschalteten Datenspeicher zu und liest nach der Ausführung des Java-Programmes die Ausgangsvariablen aus, um sie wiederum in den nachgeschalteten Datenspeicher zu schreiben .

Weiters wird als Konvention festgelegt, daß es eine Objektmethode mit fixem Bezeichner (z.B. `calculate`) gibt, deren Programmrumpf nun allerdings frei programmierbar ist.

Außerdem können anschließend weitere Objektmethoden formuliert werden.

Ein erstes Schema könnte folgendermaßen aussehen:

```
class <Funktionsbezeichner>
{
      public:
      int jUInput.i0
      float jUInput.i1
      jLPICTURE jUInput.i2

      int jUOutput.o0
      boolean jUOutput.o1

      void calculate()
      {
            ...
      }

      int function1()
      {
            ...
            return ...;
      }

      jLPICTURE function2()
      {
            ...
            return ...;
      }
}
```

Der nicht editierbare Bereich ist erstreckt sich somit von `class <Funktionsbezeichner>` bis `void calculate()` (einschließlich der Klammern die die Funktion `calculate` begrenzen). Werden im Rumpf der Funktion `calclulate` keine Anweisungen formuliert, so wird standartmäßig eine „Dummy"-Funktions bei der Gesamtübersetzung der Projektes in ein Java-Programm eingesetzt.

`Funktionsbezeichner` ist, wie der Name schon sagt, identisch mit dem im Optionenfenster festgelegten Bezeichner für die Übertragungsfunktions. In dieser Beispielskizze sind weiters die Standartbezeichner für Input (`ix`) und Output (ox) verwendet worden. Werden diese Bezeichner oder die Typen der Variablen im Optionenfenster verändert, so ändern sich automatisch allen Bezeichner und Typen der Variablen nicht nur im nicht editierbaren, sondern auch im editierbaren Bereich.

Zusammengesetzte Datentypen (wie hier im Beispiel jLPICTURE) werden durch die aktuellen Typendatenbank des Projektes verwaltet und werden bei der Übersetzung des Projektes entsprechend im Java-Code definiert.

1.7.3 Fuzzy-Units

Zur Definition der Übertragungsfunktion der Fuzzy-Units wird wie bei den Java-Units eine eigene Frame zur Verfügung gestellt. Im Untschied dazu ist es nun möglich die Fuzzy-Units, ähnlich wie bei der GPS selbst, diese graphisch zu programmieren.

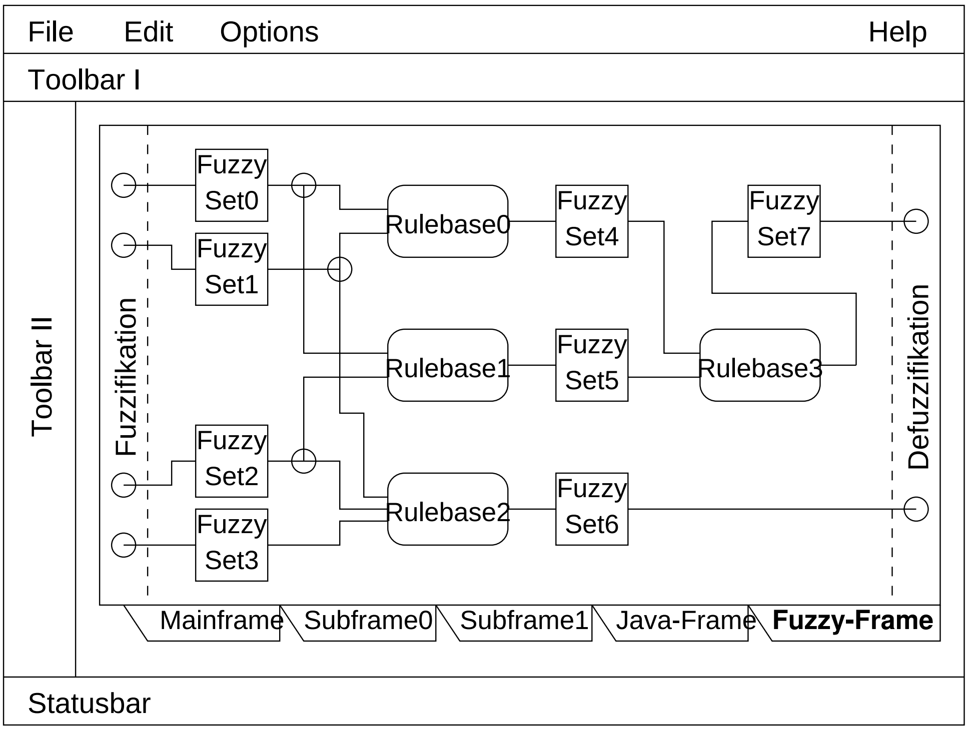

Abbildung 1.12: Beispiel eines Fuzzy-Reglers (für die Meta-Rulebase folgt ein Beispiel später).

Es werden folgende Grundelemente im Fuzzy-Frame unterschieden:

- Fuzzy-Set (die linguistischen Fuzzy-Variablen)
- Rulebase (Fuzzy-Regeln; ähnlich den Units in der GPS)
- Meta-Rulebase (Auswahl von Berechnungsergebnissen mehrerer Rulebases anhand weiterer Fuzzy-Sets)
- Connections (Verbindungen zwischen Fuzzy-Sets und Rulebases)

Ein Beispiel wie diese Komponenten graphisch realisiert werden könnten gibt Abb. 1.12.

Alle Eingangs- und Ausgangsparameter sind entweder vom Typ `integer` oder `float`. Dies kann entsprechend im Optionenfenster eingestellt werden.

Das Fuzzyfizierungs- und Defuzzyfizierungsfeld ist symbolisch zu verstehen. Natürlich findet in der Fuzzy-Set-Unit je nach dem, ob eine Rulebase folgt bzw. vorgeschaltet ist oder nicht eine Fuzzifizierung bzw. eine Defuzzyfizierung statt.

Beim Konstruieren des Fuzzy Reglers gibt es einige wichtige Regeln:

1. Nach dem Fuzzyfizierungsfeld folgt ein Fuzzy-Set.
2. Nach einem Fuzzy-Set folgt entweder eine Rule-Base oder das Defuzzifizierungsfeld.

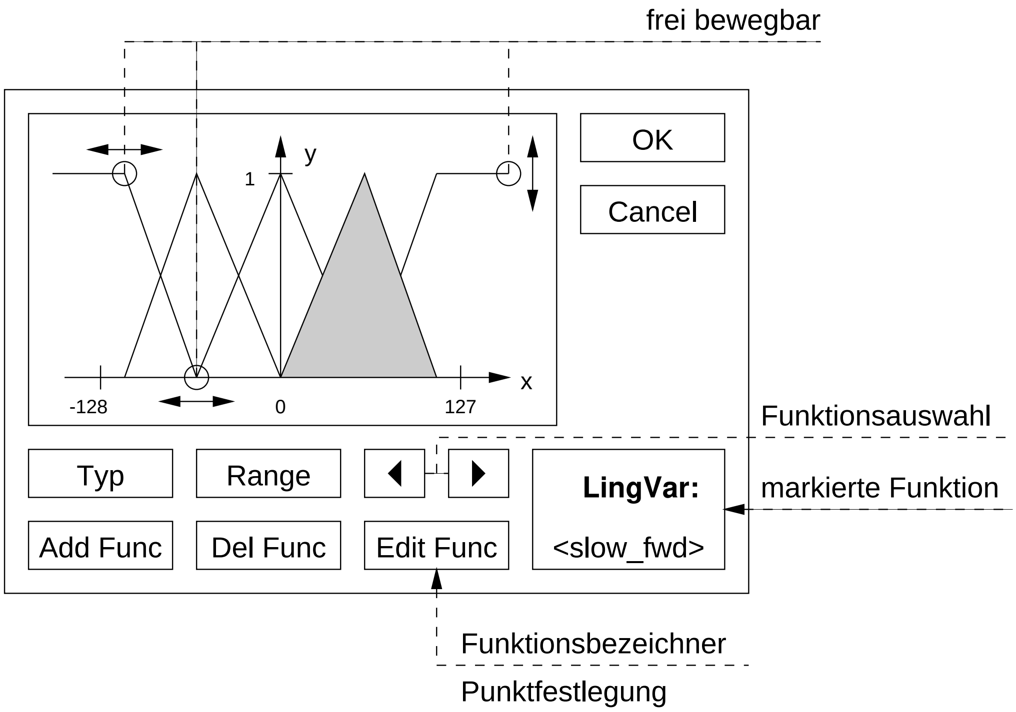

Abbildung 1.13: Beispiel eines Optionenfensters zum Erstellen von Fuzzy-Sets: Die Pfeile sollen andeuten, daß die einzelnen Punkte der Fuzzy-Funktionen mithilfe der Maus verändert werden können. Mit der Funktionsauswahl kann man eine zu editierenden Funktion auswählen. Mit `<Edit Func>` können dann Bezeichner und Punktparameter in einem weiteren Feld festgelegt werden. In Feld `LingVar` wird der linguistische Bezeichner der aktuell ausgewählten Funktion des Fuzzy-Sets angezeigt.

3. Nach einer Rule-Base folt ein Fuzzy-Set.

Die einzelnen Komponenten der Fuzzy-Frame haben wieder spezielle Eigenschaften, die im Folgenden näher erläuter werden:

Fuzzy-Set: Beim Fuzzy-Set wird ähnlich wie bei den herkömmlichen Units der GPS wieder zwischen einem Typ (das Äquivalent in der GPS wäre der Funktionstyp) und einer Instanz unterschieden. Wird eine neue Fuzzy-Set entworfen so kann man deren Typeigenschaften in einem Optionenfenster festlegen. Danach ist dieser Typ wiederum in einer Datenbank (oder Bibliothek) innerhalb des aktuellen Projektes verfügbar. Jede Instanz erhält zudem natürlich wieder einen in der jeweiligen Frame eindeutigen Bezeichner.

In dem Optionenfenster (siehe einen einfachen Entwurf in Abb. 1.13) kann nun die Fuzzy-Set anhand eines Graphen definiert werden. Dieses Optionenfenster muß folgende Funktionalität aufweisen:

- Zum einen muß festgelegt werden, um welchen numerischen Typ es sich bei der x-Achse handelt (`integer` oder `float` möglich). Bei der y-Achse handelt es sich immer um einen `float`-Wert der zwischen 0 und 1 definiert ist.

- Ein Bereich, in dem der die Funktion definiert ist muß angebbar sein.
- Weiters muß man die Anzahl der Funktionen in einem Graphen angeben bzw. immer eine neue Funktion hinzugefügt oder gelöscht werden können
- Jeder Funktion muß ein innerhalb des Fuzzy-Sets eindeutiger Bezeichner zugewiesen werden, der die linguistische Bedeutung festlegen soll.
- Es sind nur lineare Funktionen zugelassen, die jedoch an übere mehrere Punkte definiert sein kann. Daher muß auch die Anzahl der definierenden Punkte entweder dynamisch gehalten werden oder diese Anzahl muß bereits zu Beginn der Definition einer neuen Funktion angegeben werden.
- Die x- und y-Werte dieser Punkte kann entweder numerisch per Tastatur eingegeben werden oder bequem durch Ziehen der Funktion mithilfe der Maus möglich sein. Weiters sollen diese Punkte auch noch nachträglich veränderbar sein. Wesentlich dabei ist, daß das Ändern von Eigenschaften einer Instanz eines Fuzzy-Set-Typs auch dieselben Änderungen in allen bereits vorhandenen Instanzen von Fuzzy-Sets desselben Typs bewirkt.

Rulebase: Im Optionenfenster der Rulebase werden nun die Fuzzy-Regeln definiert. Es ist wesentlich, daß vorher bereits alle unmittelbar vor- und nachgeschaltenen Fuzzy-Sets fertig definiert worden sind. Es sind weiters zwei Arten von Optionenfenster zu unterscheiden:

- Zweidimensionale Tabelle
- Eine editorähnliche Eingabe zur Definition der Regeln

Erstere kann nur dann verwendet werden, falls die Rulebase höchstens zwei linguistische Eingänge besitzt. Nun können, wie in Abb. 1.14, durch Mausklicke auf die Tabellenfelder die Fuzzy-Regeln bequem definiert werden, indem jeweils eine Liste gültiger linguistischer Variablenbezeichner für die am Ausgang der Rulebase angeschlossene Fuzzy-Set erscheint, aus der der gewünschte Wert ausgewählt werden kann. Es dürfen natürlich auch Felder mit keinen Werten belegt werden.

Die zweite Möglichkeit der Regeldefinition kann unabhängig von der Anzahl der Eingänge immer verwendet werden. Dabei handelt es sich um einen Editor, dessen Eingabe jedoch nicht über die Tastatur, sondern über eine Toolbar, die sich auf der linken Seite des Fensters befindet, erfolgt. In diese Toolbar müssen folgende Sprachkonstrukte vorhanden sein:

- `AND, OR`
- `THEN`
- sowie zwei Listen über die jeweils
  - das entsprechende **Fuzzy-Set**, welches am Eingang bzw. Ausgang der Rulebase angeschlossen ist,

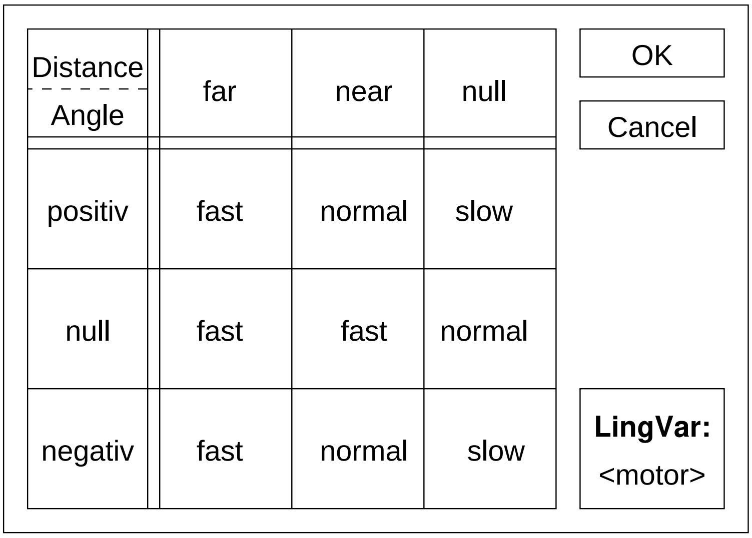

Abbildung 1.14: Zweidimensionale Tabelle zum Erstellen der Fuzzy-Regeln. Diese Form ist jedoch auf max. zwei linguistische Variablen als Eingang und auf die Verwendung des AND-Operators beschränkt (häufigster Anwendungsfall).

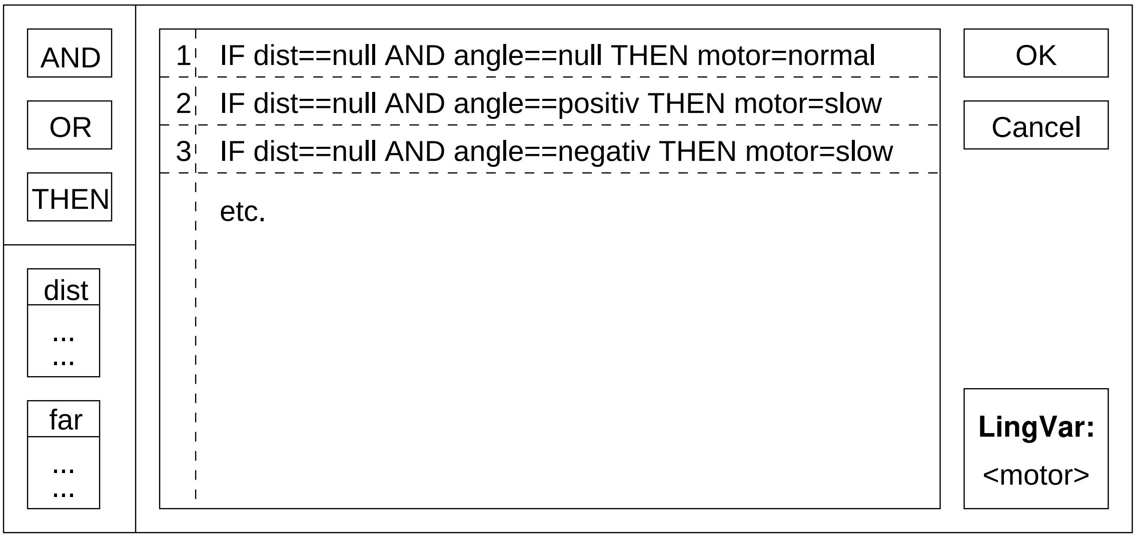

Abbildung 1.15: Editorähnliche Definition der Regeln ohne Einschränkungen.

– und die gewünschte **linguistische Variable** des jeweiligen Fuzzy-Sets ausgewählt werden können.

In jeder Zeile kann nun eine Fuzzy Regel definiert werden (siehe dazu Abb. 1.15). Im Endprogramm soll noch zusätzlich eine Gewichtung jeder einzelnen Regel möglich sein.

Meta-Rulebase: Mithilfe dieser Art von Rulebase können mehrere Berechnungsergebnisse anderer (normativen) Rulbases nocheinmal einer Fuzzy-Regelung unterworfen werden, die wiederum von Fuzzy-Sets abhängen.

Die Eingänge der Meta-Rulebases sind unterteilt in:

- Herkömmliche Fuzzy-Set Eingänge und
- Gruppen von Fuzzy-Set Eingängen, wobei jede Gruppe in Anzahl und Typ der vorgeschalteten Fuzzy-Sets übereinstimmen muß.

Der Ausgang ist vom Typ her wie eine einzelne Gruppe vom Eingang definiert. Sobald eine Eingangsgruppe angeschlossen wurde sind somit sowohl die anderen Eingangsgruppen typmäßig definiert, als auch der Ausgang fix festgelegt. Es kann bei Belegung des Ausganges kein neuer Typ erzeugt werden und aus der Datenbank für Fuzzy-Sets kann nur der mit der Eingangsgruppe identische Fuzzy-Set ausgewählt werden.

Eine graphische Darstellung findet sich in Abb. 1.16.

Das Erstellen der Regeln innerhalb der Meta-Rulebase funktioniert prinzipiell genauso wie in Abb. 1.14 und Abb.1.15 dargestellt. Als Input werden dabei jedoch nur die Nicht-Gruppen Eingänge herangezogen. Je nach dem zu welchem Prozentsatz nun diese Regeln erfüllt sind, wird nun die Werte der einzelnen Gruppen gemittelt und am Ausgang weitergegeben. Als Folgerung der einzelnen Regeln ist also nicht ein lingustischer Wert (wie z.B. „far“, „slow“, etc.) einer bestimmten linguistischen Variable (wie z.B. „dist“, „speed“, etc.) anzugeben, sondern nur welche Gruppe im Falle des Zutreffens einer Regel zur Berechnung des Ausgangswertes herangezogen werden soll.

1.7.4 Net-Units

Für Net-Units sollen bereits fertig konstruierte und trainierte Neuronale Netze importiert werden können. Vorzugsweise wird dabei auf den Stuttgarter Neuronale Netze Simulator (SNNS) zurückgegriffen. Es ist zu diskutieren, welche Arten von Netze importiert werden können.

Es soll weiters ein Programm entworfen werden, daß ähnlich wie das zu dem Softwarepaket des SNNS zugehörigen Konvertierungsprogramm SNNS2C (stimmt das?) aufgebaut sein soll und somit Neuronale Netze in ein Java-Programm umwandeln kann.

Für die Eingangstypen dieser Units sind nur atomare Datentypen erlaubt, welche aber bereits bei der Importierung des Netzes festgelegt werden. Der Programmierer soll möglichst wenig Eingriffsmöglichkeiten in das Neuronale Netz selbst haben.

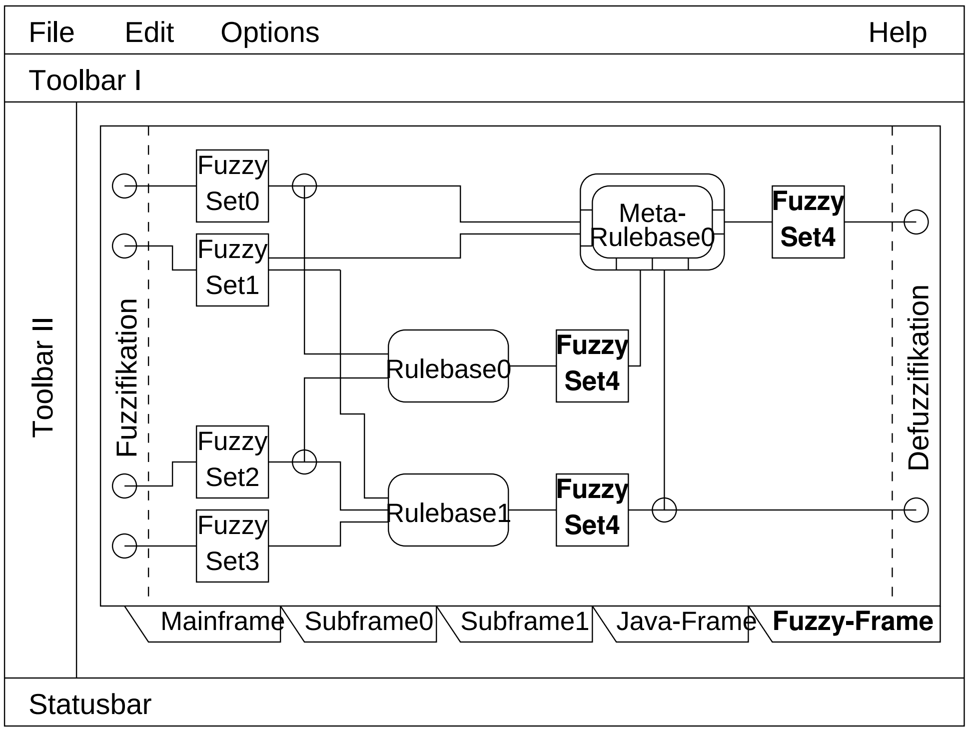

Abbildung 1.16: Beispiel einer Verschaltung eines Fuzzy-Reglers unter Verwendung einer Meta-Rulebase.

1.8 Schnittstellen zu DIMAS und EMMA

Folgt in der nächsten Version dieses Dokumentes.

Anhang A

Besprechungsprotokolle:

A.1 Wintersemester 1999/2000:

In diesem Semester wurden erste Grundkonzepte in verschiedenen Besprechungen erörtert. Dazu gehörten:

- Aufgabenverteilung der einzelnen Programme des Projektes „RoboSim“ an einzelnen Personen
- Grobe Klärung der Schnittstellen zwischen diesen Programmen
- Absteckung des Funktionsumfangs der einzelnen Programme
- Entwicklung erster Grundkonzepte und innerer Strukturen dieser Programme

An dem Projekt beteiligte Personen:

- Roland Schweiger
- Helmut Mayer
- Katstaller Gerhard
- Schmidbauer Josef
- Wiesbauer Gerald
- Stieglbauer Gerald

Anhang B

Dokumentversionen:

B.1 Version 0.1 (10.03.2000):

In dieser Version wurden folgende Kapitel neu geschrieben:

- Erste Fassung der Kapitel bis einschließlich 1.5.
- Erste Konzeption des Kapitels 1.6 anhand der if-Kontrollstruktur.

B.2 Version 0.2 (17.03.2000):

In dieser Version wurde das Dokument um folgende Punkte erweitert:

- Überarbeitung und Ergänzung der Texte bis einschließlich 1.5.
- Neufassung des Kapitels 1.6, insbesondere des Unterpunktes 1.6.2.
- Hinzufügen der Unterpunkte 1.6.1 und 1.6.3.

B.3 Version 0.3 (24.03.2000):

In dieser Version wurde das Dokument um folgende Punkte erweitert:

- Überarbeitung des Unterpunktes 1.6.3 und Hinzufügen des Algorithmus' 4.
- Hinzufügen der Unterpunkte 1.6.4 und 1.6.5.

B.4 Version 0.4 (04.04.2000):

In dieser Verson wurde das Dokument um folgende Punkte erweitert:

- Hinzufügen des Kapitels 1.7.