# Entwurf und Implementation einer universellen Datenschnittstelle für eine Robotersteuerung auf Basis standardisierter Übertragungsprotokolle

**Diplomarbeit**

im Fachbereich Automatisierungstechnik

*von*

**Sören Zimmermann**

*eingereicht am*

**2003-08-20**

*bei*

Prof. Dr.-Ing. Dipl.-Inform. Eckhard C. Bollow

Prof. Dr. rer. nat. Dipl.-Inform. Helmut Faasch

## Kurzfassung der Diplomarbeit

Im Rahmen dieser Arbeit sollte für eine Robotersteuerung eine neue, möglichst universelle Datenschnittstelle entworfen und implementiert werden. Diese Datenschnittstelle sollte in der Lage sein, beliebige, strukturierte Daten im Textformat auszutauschen.

Ein standardisiertes Format, das alle definierten Bedingungen erfüllt, ist XML. Aus diesem Grund wurde eine XML-Schnittstelle in das Realtime-Betriebssystem VxWorks implementiert, das auf der Hardware der Robotersteuerung lauffähig ist.

Um diese Schnittstelle zu verwenden, wurde das bisherige Datenformat zur Übertragung von Roboterkonstanten überarbeitet und in XML neu definiert.

Um mit dem neuen Konstantenformat arbeiten zu können, wurde ebenfalls ein Konstanteneditor entwickelt, der unter Microsoft Windows lauffähig ist und die XML-Konstantendaten lesen, bearbeiten und erzeugen kann.

## Abstract

Within the scope of this diploma thesis a new interface for data exchange for a robot control should be designed and implemented. This interface should be able to exchange any structured data using text files.

A standard format meeting all the defined requirements is XML. For this reason a XML interface has been implemented into the real-time operating system VxWorks. This interface runs on the hardware platform of the robot control.

To use the new interface the former data exchange format of the robot constants has been redesigned and defined in XML.

To work with the new robot constants format an editor has been developed which runs under Microsoft Windows. This editor reads, edits and writes the new XML constants format.

# Inhaltsverzeichnis

# 1 Einleitung

Unter einem Industrieroboter versteht man im technischen Sinne eine Maschine[(a)] zur Handhabung[(b)] von Werkzeugen oder Werkstücken.

Industrieroboter sind heute ein wichtiges Werkzeug bei der automatisierten Produktion. Ihre Entwicklung begann bereits Ende der 70er Jahre des vorigen Jahrhunderts. Seitdem wurde der Industrieroboter ständig verbessert und stets auf dem aktuellen Stand der Technik gehalten. Da es sich bei einem Roboter, insbesondere bei der Robotersteuerung, um ein sehr komplexes System handelt, kann dieses jedoch oftmals nicht vollständig an neue Technologien und Techniken angepasst werden. Somit basiert eine Robotersteuerung im wesentlichen auf einem langjährig entstandenen und erprobten Kern.

Auch die Robotersteuerung S.RS2 aus dem Hause der SEF Roboter GmbH, 21379 Scharnebeck, ist Ergebnis eines solchen langen Entwicklungsprozesses. Zu der Zeit, als einige der Basisfunktionen dieser Steuerung entstanden sind, war Rechenleistung teuer und Speicherplatz knapp bemessen. Dieses hat zu proprietären, teilweise minimalistischen Datenformaten geführt, mit denen die Steuerung Informationen mit der Außenwelt austauscht.

Wurden Robotersysteme zunächst einzeln eingesetzt und programmiert, entstand bald die Notwendigkeit, diese miteinander zu vernetzen. Deshalb wurden industrielle Standardschnittstellen und -protokolle, z.B. der Interbus [1.3], in die Steuerung implementiert, damit die Robotersysteme untereinander, aber auch mit fremden Systemen, kommunizieren konnten.

Heute ist die Vernetzung der Produktion weit fortgeschritten. Die Produktionssysteme werden nicht mehr nur untereinander, sondern auch mit Warenwirtschaftssystemen vernetzt und sogar über das Internet gewartet und gesteuert.

Aus diesem Grund ist es wünschenswert, das Datenaustauschformat so zu gestalten, dass es standardisiert und somit auch für fremde Systeme lesbar ist. Ein solches Format für den Austausch beliebiger, strukturierter Daten findet sich in der vom W3C[(c)] definierten XML - der eXtensible Markup Language.

(a) »Eine Maschine ist eine Gesamtheit von miteinander verbundenen Teilen oder Vorrichtungen, von denen mindestens eines beweglich ist, sowie gegebenenfalls von Betätigungsgeräten, Steuer- und Energiekreisen usw., die für eine bestimmte Anwendung, wie die Verarbeitung, die Behandlung , die Fortbewegung und die Aufbereitung eines Werkstoffes zusammengefügt sind«. Zitiert nach [1.1].

(b) »Handhaben ist das Schaffen, definierte Verändern oder vorübergehende Aufrechterhalten einer vorgegebenen räumlichen Anordnung von geometrisch bestimmten Körpern in einem Bezugskoordinatensystem. Es können weitere Bedingungen - wie z.B. Zeit, Menge und Bewegungsbahn - vorgegeben sein.« Zitiert nach [1.2].

(c) W3C: World Wide Web Consortium. Ein internationales, unabhängiges Gremium, das Standards für das Internet (WWW) definiert, z.B. das HTML-Protokoll zur Übertragung von Internetinhalten. [1.4]

Die Grundidee liegt nun darin, in die Robotersteuerung S.RS2 eine XML-Schnittstelle zu implementieren, damit dieses Datenformat gelesen und geschrieben werden kann.

Zudem müssen die zu übertragenden Daten formal definiert werden, damit sie den XML-Kriterien genügen.

Exemplarisch wurde dies im Rahmen der vorliegenden Arbeit für die sogenannten Roboter-Konstanten vorgenommen.

Dabei wurde das bisherige Konstantenformat überarbeitet und eine Reihe weiterer Forderungen an dieses Format definiert und eingehalten.

Um den Nutzen des neuen Konstantenformates deutlich zu machen, wurde ebenfalls ein Konstanten-Editor entwickelt, der dieses Format lesen, bearbeiten und erzeugen kann. Der Konstanteneditor ist unter Microsoft Windows[(a)] lauffähig und demonstriert somit, wie Daten der Steuerung mit Hilfe des XML-Formates auf einem beliebigen System genutzt werden können.

(a) Die Wiedergabe von Gebrauchsnamen, Handelsnamen, Warenbezeichnungen usw. in dieser Arbeit berechtigt auch ohne besondere Kennzeichnung nicht zu der Annahme, dass solche Namen im Sinne der Warenzeichen- und Markenschutz-Gesetzgebung als frei zu betrachten wären und von daher von jedermann benutzt werden dürften.

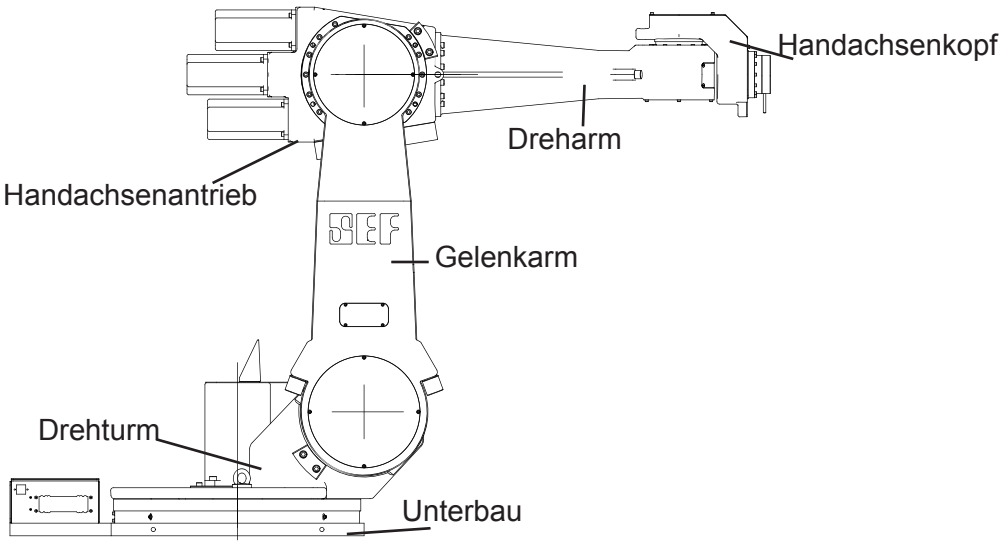
# 2 Robotersysteme

Ein Robotersystem besteht aus zwei wesentlichen Komponenten: Der Mechanik und der Steuerung. Um ein Robotersystem betreiben zu können, müssen diese beiden Elemente aufeinander abgestimmt werden.

## 2.1 Die Robotermechanik

Eine Robotermechanik besteht aus der Aneinanderreihung von Armteilen, die durch Gelenke (Achsen) miteinander verbunden sind. Die Mechanik umfasst alle Elemente, die notwendig sind, den Roboter zu bewegen, wie Motoren, Getriebe, Lager und Übertragungselemente wie Riemen oder Ketten.

Typischerweise werden Robotermechaniken mit 6 Achsen ausgeführt, man spricht dann von einem 6-Achs-Knickarmroboter. In der folgenden Abbildung ist eine derartige Bauform schematisch dargestellt.

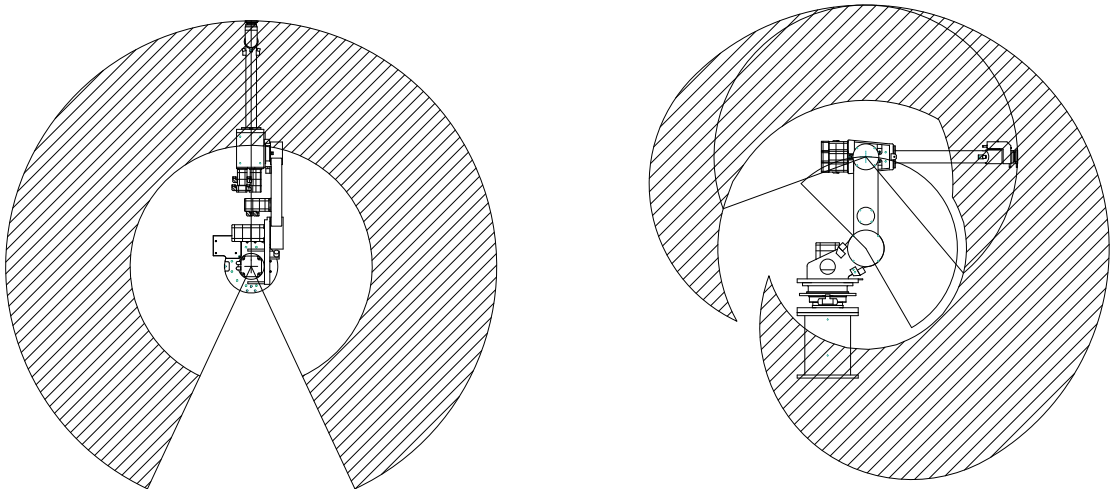



Darstellung eines 6-Achs-Knickarmroboters

Diese Konstruktion hat den Vorteil, dass innerhalb des Arbeitsraumes der Mechanik zwei 3-dimensionale Koordinatensysteme überlagert werden können. Das erste 3D-Koordinatensystem bestimmt die Lage im Arbeitsraum, das zweite Koordinatensystem die Orientierung des Werkzeugs. Man sagt, dass eine solche Konstruktion den Freiheitsgrad $f = 6$ besitzt.

Roboter, die mit weniger Achsen konstruiert werden, besitzen entsprechend einen geringeren Freiheitsgrad. Eine weit verbreitete Bauform ist hier der SCARA-Roboter. Die Abkürzung SCARA steht für »Selective Compliance Assembly Robot Arm«, zu deutsch »Schwenkarm-Roboter«. Diese Modelle besitzen meist einen Freiheitsgrad von $f$=4. Ihre räumliche Bewegung ist eingeschränkt. Sie eignen sich für Arbeiten, bei denen sich die Werkzeugspitze, der sogenannte Tool Center Point (TCP), stets senkrecht zur Arbeitsebene befindet, also z.B. beim Bohren oder Palettieren.

Bei einer 6-Achs-Knickarmkinematik kann das Werkzeug frei im Raum positioniert werden. Der Arbeitsraum ist idealisiert kugelförmig um die Robotermechanik angeordnet. Es ist jedoch zu beachten, das der Bereich, den der TCP nicht nur erreichen, sondern in dem er auch orientiert werden kann, nur einen Teilbereich des gesamten Arbeitsraums ausmacht [2.1].



Arbeitsraum eines 6-Achs-Knickarmroboters

## 2.2 Die Steuerung

Die Steuerung für ein Robotersystem umfasst die elektrischen und elektronischen Elemente, die notwendig sind, um die Robotermechanik zu bewegen.

### 2.2.1 DAS PROGRAMMIERHANDGERÄT

Das Programmierhandgerät, kurz PHG genannt, stellt die Schnittstelle zwischen dem Benutzer und der eigentlichen Rechnereinheit dar. Über das Programmierhandgerät werden während der Einrichtung des Roboters alle Funktionen zur Verfügung gestellt, die notwendig sind, ein Programm zu erstellen. Dazu zählt nicht nur das Verfahren der Roboterachsen, sondern auch die Verwaltung von E/A-Modulen, häufig eine SPS, Werkzeugbefehle oder auch die Eingabe von beschreibendem Freitext.

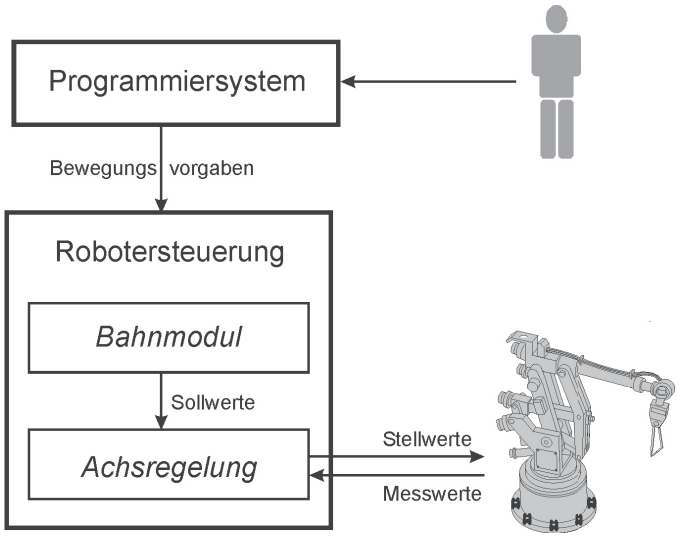
Im Automatikmodus des Roboters können über das PHG Status- und Fehlermeldungen angezeigt und ggf. quittiert werden. Das PHG ist, obwohl es die Schnittstelle zur Rechnereinheit bildet, meistens ein eigenständiges Gerät. In dem Falle der S.RS2 läuft auf dem PHG sogar ein anderes Betriebssystem als auf der Steuerung an sich.

### 2.2.2 DIE RECHNEREINHEIT

Die Rechnereinheit bildet das Herzstück der Robotersteuerung. Sie setzt sich aus einem Hardware- und einem Softwareanteil zusammen. Die Hardware besteht üblicherweise aus einem Prozessorsystem, Schaltungen für personelle und maschinelle Sicherheit, analogen und/oder digitalen E/A-Modulen und optionalen Feldbusanbindungen.

Der Softwareanteil läuft in jedem Fall unter einem Echtzeit-Betriebssystem und ist für die Regelung der einzelnen Achsen, die Überwachung der Sicherheitsfunktionen und permanente Selbstdiagnose zuständig. Die Selbstdiagnose achtet z.B. darauf, ob sich der Roboter auf der berechneten Bahn befindet, dass bestimmte Softwaregrenzen nicht überschritten werden und ob die Kommunikation mit den Antrieben und dem PHG einwandfrei funktioniert.

Darüber hinaus stellt die Software ein Programmiersystem zur Verfügung, über welches ein Benutzer den Roboter einrichten kann:




Programmiersystem als Bedienerschnittstelle

Über das Programmiersystem werden der Steuerung Bewegungsvorgaben eingegeben, z.B. eine Menge von Raumpunkten, die in einer bestimmten Reihenfolge abzufahren sind. Diese werden in dem Bahnmodul, der Bewegungssteuerung, untersucht. Durch Interpolation zwischen diesen Punkten wird eine geeignete Roboterbahn berechnet. Diese führt dann zu Sollwerten, die in der Achsregelung zusammen mit den rückgekoppelten Messwerten dafür sorgen, dass die einzelnen Motoren des Roboters korrekt geregelt werden können, so dass der Roboter schließlich die gewünschten Bewegungen vollführt.

Das Programmiersystem ist die zentrale Schnittstelle zwischen dem Benutzer und der Funktionalität der Steuerung. Ihr Aufbau und Bedienung bestimmen, wie effizient der Roboter letztendlich programmiert werden kann. Üblicherweise wird das Programmiersystem über das PHG bedient.

### 2.2.3 DIE ANTRIEBSEINHEIT

Um die von der Recheneinheit ermittelte Bahn auch in mechanische Bewegung umzusetzen, müssen die Motoren der Mechanik entsprechend angesteuert werden. Je nach Bauform und Leistung der Mechanik können dazu große elektrische Leistungen vonnöten sein. 6-Achs-Knickarmkinematiken können so ausgelegt sein, dass sie 500 kg oder mehr bewegen können.


Mechanik mit 500 kg Traglast (Werkbild KUKA)

Kinematiken für hohe Lasten werden häufig mit Drehstrom-Servomotoren ausgerüstet. Um diese Motoren ansprechen zu können, werden entsprechend der Zahl der Achsen Servo-Verstärker sowie deren Spannungsversorgung benötigt. Neben der reinen Verstärkung können moderne Servo-Verstärker auch bereits die Logik für den Achsregler enthalten.

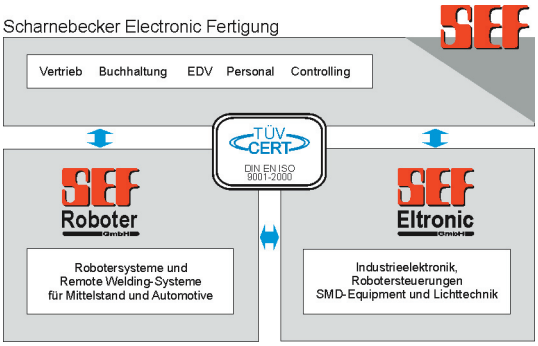
# 3 Die Robotersteuerung S.RS2

Die Robotersteuerung S.RS2 ist eine industrielle Robotersteuerung, die alle in Kapitel 2 beschriebenen Merkmale aufweist. Die S.RS2 ist ein Produkt der SEF Roboter GmbH.

## 3.1 Die SEF Roboter GmbH

Die SEF Roboter GmbH ist eine Tochtergesellschaft der SEF GmbH, Scharnebecker Electronic Fertigung mit Sitz in Scharnebeck. [3.1]

Die Konzernmutter SEF GmbH beschäftigt derzeit ca. 100 Mitarbeiter, davon 30 in der SEF Roboter GmbH und 55 in der SEF Eltronic GmbH.




SEF-Gruppe in der Übersicht

Die SEF GmbH besitzt bereits seit Anfang der 80er Jahre Erfahrungen bei der Entwicklung von Robotersteuerungen. Nach der Lizenzübernahme einer Robotermechanik von einem großen Automobilkonzern gründete die SEF GmbH 1996 ihre Tochter SEF Roboter GmbH. Seitdem bietet die SEF Roboter komplette Robotersysteme für die Automobilindustrie und den Mittelstand an.

Die zu den Robotersystemen gehörenden Steuerungen S.RS2 werden bei der SEF Eltronic GmbH gefertigt und von der SEF Roboter GmbH mit den Mechaniken zu einem Gesamtsystem zusammengefügt. Die Software der S.RS2 wird zur Zeit im Auftrag der SEF Roboter GmbH von der ADC GmbH [3.2] entwickelt bzw. weiterentwickelt.

## 3.2 Der Schaltschrank

### 3.2.1 Die Hardware

Robotersteuerung S.RS2

Die Hardware der S.RS2 besteht aus vier Hauptgruppen:

- Rechnereinheit
- Servoeinheit
- Schrankausrüstung
- Programmierhandgerät

Die Rechnereinheit besteht aus folgenden Elementen:

- PC-Einschub basierend auf PCI-Bus mit Intel Pentium 266 MHz CPU mit zwei Ethernet-Schnittstellen (für PHG und Netzwerkanbindung)
- RCC Robot Communication Controller als Koppler zwischen PCI-Bus und Servolink
- Spannungsausfallüberbrückung (sicheres Herunterfahren bei Spannungsausfall)
- SRK (Systemrelaiskarte) und SLK (Systemlogikkarte) für die personellen und maschinellen Sicherheitsfunktionen
- optionalen SPS-Modul mit 16 oder 32 Ein- und Ausgängen
- DA/AD-Modul (Digital-Analog/Analog-Digital-Modul), optional mit 2 analogen Ein- bzw. Ausgängen
- optionaler Feldbusanbindung mit Interbus oder Profibus

Die Servoeinheit besteht aus folgenden Komponenten:

- Powermodul für die Servoendstufen
- Servoendstufen (6 Achsen standardmäßig, optional bis zu 12 Achsen ansteuerbar)
- Backpanel zur Verbindung der Servoendstufen
- Ballastwiderstand (zum Abbau überschüssiger Energie aus den Antrieben)

Die Schrankaufrüstung setzt sich aus folgenden Elementen zusammen:

- Netzteil für interne Gleichspannungsversorgung
- Netzsicherung und Netzfilter
- KC2-Modul (Leistungsschütze zum Schalten der Antriebsenergie)
- Servicesteckdose 230 V
- Freie Tragschiene für weitere Optionen
- Zweikreis-Kühlsystem mit optionalem Klimagerät
- Steckerblech für äußere Anschlüsse

Über den Servolink-Bus erfolgt die Kommunikation zwischen der CPU und der Roboterperipherie wie z.B. den Servoverstärkern und den Sicherheitskreisen. Bis zu 20 Teilnehmer können, über eine Strecke bis zu 100 Metern, eingebunden und deren Eigenschaften frei programmiert werden.

### 3.2.2 DIE SOFTWARE

Als Betriebssystem kommt das industrielle Real-Time-System VxWorks von WindRiver [3.3] zum Einsatz. VxWorks ist das weltweit führende Betriebssystem für zeit- und sicherheitskritische Anwendungen.

Einige wichtige Module der S.RS2 sind

- Sensorschnittstelle: Ermöglicht eine Echtzeitkorrektur der Roboterbahn in Abhängigkeit von Sensorinformationen
- Konfigurationsschnittstelle: Austausch von Konfigurationsdaten wie z.B. Konstanten, SPS-Programmen oder Folgen.
- Netzwerkmodul: Netzwerkkommunikation zu einem Prozess- oder Leitrechner.
- Benutzerschnittstelle: Schnittstelle zum Bedienen und Konfigurieren des Systems über das Programmierhandgerät.
- Bahnsteuerung: Die Bahnsteuerung ermöglicht verschiedene Interpolationsarten zwischen zwei anzufahrenden Raumpunkten, wie z.B. die Point-to-Point Bewegung (PTP) oder lineare oder zirkuläre Bewegungen.

## 3.3 Bedienung und Programmierung

Die gesamte Bedienung und Programmierung erfolgt standardmäßig über das Programmierhandgerät. Das PHG ist ein eigenständiges, vom Schaltschrank unabhängiges Gerät. Das Betriebssystem des PHGs ist Linux.

Das PHG verfügt neben den wichtigsten Systemfunktionstasten über eine numerische Tastatur, einen Not-Aus-Schlagtaster, einen Zustimmtaster und einen Joystick.

Über das PHG stehen drei Betriebsarten zur Verfügung:

- Hand: Verfahren der einzelnen Achsen, Programmieren des Systems und Einstellen einzelner Parameter
- Single-Step: Punktweises Abfahren von programmierten Folgen (vorwärts und rückwärts)
- Automatik: Automatischer Ablauf von programmierten Folgen

Wenn der Roboter über das PHG programmiert wird, erfolgt dieses im Teach-In Verfahren: Der Roboter wird an die Positionen gefahren, die später Punkte einer Folge werden sollen. Diese werden dann übernommen und können mit zusätzlichen Informationen, wie SPS-Bedingungen oder Interpolationsart, hinterlegt werden.

Alternativ zu dem Teach-In kann die Folge in einem Simulationsprogramm erstellt und in die Steuerung heruntergeladen werden. Diese Art der Programmierung bezeichnet man als Offline-Programmierung.

Die S.RS2 kann bis zu 127 Folgen mit je 1500 Punkten verwalten.

# 4 Die Konfigurationsschnittstelle

## 4.1 Beschreibung der Schnittstelle

Die Konfigurationsschnittstelle ist derjenige Softwareteil, über den die Robotersteuerung Daten austauscht, die keine Echtzeit benötigen. Wenn eine neue Folge oder ein neuer Satz Konstanten geladen wird, ist dieses nicht zeitkritisch.

Die wesentlichen Informationen, die über die Konfigurationsschnittstelle übertragen werden, sind Folgen, SPS-Programme und die Konstanten.

Alle Daten, die über diese Schnittstelle übertragen werden, sind in einem eigenen, proprietären Format gespeichert. Diese Formate orientieren sich an der Datenstruktur, in der die Daten innerhalb der Steuerung gespeichert sind. Diese Formate werden in reinen Textdateien, in denen die Informationen klar lesbar und frei editierbar abgespeichert werden, übertragen.

Der Aufbau der Textdateien hingegen ist fest vorgegeben und daher unflexibel. Diese Textdateien besitzen keine einheitliche Struktur und können keinen erklärenden Kommentar beinhalten. Die Definitionen der Textdateien sind nur innerhalb der Entwicklungsdokumentation zu finden und für Außenstehende somit nicht einsichtig. Zudem sind die Einträge in den Textdateien in der Regel keiner Konstanten zuordbar, da sie als reiner Wert, ohne Bezug zu einer Variablen abgelegt sind.

Natürlich ergeben sich aus einem solchen Format auch Vorteile:

- Minimaler Platzbedarf
- Schnelle Verarbeitung innerhalb der Steuerung, da genau definiert ist, in welchen Zeichen welche Information hinterlegt ist
- Sprachenunabhängigkeit, da keine Probleme bei den Zuordnungen auftreten können, wie z.B. bei der Verwendung von »`Achse`« anstelle von »`axis`«.

Durch aktuelle Rechengeschwindigkeiten und hohe Speicherkapazitäten stellen die beiden ersten Punkte der vorstehenden Aufzählung heute jedoch keinen besonders großen Vorteil mehr da. Lediglich das Problem der Sprachenunabhängigkeit muss weiterhin beachtet werden, insbesondere da die SEF Roboter GmbH ein weltweit operierendes Unternehmen ist, deren Robotersysteme bereits in Spanien, Mexiko, Brasilien oder China eingesetzt werden.

Insgesamt ist es jedoch wünschenswert, die bisherigen Datenformate durch ein einziges, standardisiertes Format abzulösen.

Eine wichtige Forderung, die an ein neues Format gestellt wird, ist, dass es sich ebenfalls um ein reines Textformat handeln muss, damit eine einfache Ansicht oder Änderung weiterhin möglich bleibt. Das Datenformat muss ferner strukturiert

und lesbar sein, die Zuordnungen der Werte zu den Konstanten erlauben und Kommentare enthalten können.

Ein Format, dessen Eigenschaften diese Forderungen erfüllt, ist das XML-Format. Es wird in Kapitel 5 ausführlicher erläutert.

Da es sich bei dem XML-Format um einen offenen Standard handelt, können die Daten, die in diesem Format hinterlegt sind, von ebenfalls standardisierten Werkzeugen verarbeitet werden. Die Definition des Datenformates läge allen offen, die sich mit der Steuerung beschäftigen. Zudem kann das XML-Format für alle Daten, die über die Konfigurationsschnittstelle ausgetauscht werden, verwendet werden. Außerdem ist das XML-Format offen für Erweiterungen, die zukünftig folgen könnten bei gleichzeitiger Abwärtskompatibilität.

Diese Eigenschaften haben dazu geführt, dass in die S.RS2 eine XML-Schnittstelle implementiert werden sollte. Eine reine Implementation reicht jedoch nicht aus, es müssen die zu übertragenden XML-Daten formal definiert und die Definition in der Steuerung hinterlegt werden. Zudem müssen die Funktionen, die zum Lesen und Schreiben der bisherigen Datenformate vorhanden sind, überarbeitet werden. Im Rahmen der vorliegenden Arbeit habe ich mich auf die Umstellung der Konstanten-Daten auf das neue XML-Format beschränkt.

## 4.2 Roboterkonstanten

### 4.2.1 Allgemeines

Die Roboterkonstanten sind diejenigen Werte, die, bezogen auf einen bestimmten Schaltschrank in Verbindung mit einer bestimmten Mechanik, das Gesamtsystem beschreiben bzw. das Verhalten des Gesamtsystems festlegen.

Zu diesen Werten gehören:

- Steuerungscharakteristik
- Robotermechanik
- Ein- und Ausgabehardware
- Werkzeuge
- Nachladbare Zusatzmodule
- Programmierung der freilaufenden SPS

Die Konstanten werden dauerhaft in der Steuerung gespeichert und bei jedem Systemstart automatisch initialisiert.

Während ein Teil der Konstanten, z.B. die der Robotermechanik, selten oder nie geändert werden müssen, dienen ein Großteil der Konstanten der optimalen Anpassung des Systems an eine installierte Applikation.

Die Konstanten können entweder über das PHG direkt in der Steuerung geändert oder über Konstantendateien eingeladen werden. Einige Systemkonstanten sind

gegen unbeabsichtigtes oder unbefugtes Ändern geschützt. Um sie verändern zu können, muss sich der Benutzer bei dem System authentifizieren, z.B. mittels einer Schlüsseldiskette. Welche Konstanten auf diese Weise geschützt sind, ist in der Robotersteuerung hinterlegt.

### 4.2.2 Das Konstantenformat der S.RS2

Bei der S.RS2 können Konstanten aus mehreren Werten bestehen. Einer Konstanten wird immer eine Nummer zugeordnet. Ein Beispiel:

Die Konstante 203 beschreibt die Achslängen der Mechanik in Millimetern. Insgesamt besteht die Konstante 203 aus 6 Werten, für 6 mögliche Achsen.

Mit dem bisherigen Format werden die Roboterkonstanten auf drei Dateien aufgeteilt:

| Dateiname | Inhalt |
|---|---|
| WERKDAT.SAV | Werkzeugkonstanten für 7 Werkzeuge |
| ROBDATEN.SAV | Spezielle Konstanten für einen Robotertyp |
| KONSTDAT.SAV | Sonstige Konstanten, insbesondere Regler-, E/A- und Bahnmodulkonstanten |

Dateiaufteilung der S.RS2-Konstanten

Dabei ist die Zuordnung der Konstanten nicht immer eindeutig. Durch diese Trennung kommt es unter anderem vor, dass die Konstante für die maximale Achsgeschwindigkeiten sowohl in der Datei für die Roboter- als auch der für die sonstigen Konstanten abgelegt ist. Dies führt zu unerwünschten Effekten, falls die Konstanten in beiden Dateien ungleiche Werte besitzen.

Jede Zeile einer Konstantendatei entspricht dabei einem Konstantenwert. Daher dürfen keine Zeilen vertauscht oder zwischengefügt werden. Eine Erweiterung des Formates ist nur möglich, indem Zeilen am Ende der Datei angehängt werden. Dies ist in der Vergangenheit mehrfach geschehen, so dass thematisch zusammenhängende Einträge, z.B. Werte für einzelne Achsen ein und derselben Konstanten, an unterschiedlichen Stellen innerhalb einer Datei zu finden sind.

Exemplarisch ist in der folgenden Tabelle die Definition der Konstantendatei »ROBDATEN.SAV« dargestellt, da sie die kleinste der Konstantendateien darstellt. Links neben der Tabelle befindet sich der Beginn des Inhalts solch einer Roboter-konstanten-Datei.

Die Definition der »KONSTDAT.SAV« ist weitaus umfangreicher, sie enthält 1.985 Einträge. Die Datei »WERKDAT.SAV« enthält momentan 265 Einträge.

Genaue Beschreibungen der einzelnen Konstanten kann man dem Programmierhandbuch der Robotersteuerung entnehmen. [4.1]

```
ETYP
68
4
1600
600
1000
0
0
0
0
7a1200
7a1200
7a1200
7a1200
7a1200
7a1200
80000
80000
80000
80000
80000
80000
8090672
7903706
7533875
6915625
7445334
6728034
1
1
1
1
1
1
8178490
8080157
9685125
7991376
8554666
9314365
968732
968732
968732
968732
968732
968732
-10.000000
10.000000
-10.000000
10.000000
-10.000000
10.000000
262144.000000
262144.000000
1125000.000000
1125000.000000
253788.000000
```

Beginn einer Roboter-Konstantendatei

| Zeilen | Inhalt | Datentyp |
|---|---|---|
| 1 | Robotername (max. 7 Zeichen) | `char[8]` |
| 2 | Robotertyp | `int` |
| 3 | Anzahl der Achsen | `int` |
| 4 - 9 | Achslängen [mm] | `int` |
| 10 | Keine Bedeutung | |
| 11-22 | Winkelkodierer-Grundeinstellung [Bit] | `long` |
| 23 - 34 | Untere Softwaregrenze [Bit] | `long` |
| 35 - 46 | Obere Softwaregrenze [Bit] | `long` |
| 47 - 52 | Begrenzung des Arbeitsraums [m] | `float` |
| 53 - 64 | Übersetzungsverhältnis [Bit/90°] | `float` |
| 65 - 76 | Offset für Normierung [Grad] | `float` |
| 77 - 88 | Achsgeschwindigkeit [Grad/s] | `float` |
| 89 - 100 | Achsgeschwindigkeit in Hand [%] | `float` |
| 101 - 112 | Achsbeschleunigung [Grad/s²] | `float` |
| 113 | ZTF-Modul aktiv/passiv | `int` |
| 114 - 123 | ZTF-Konstanten | `int` |
| 124 | Keine Bedeutung | |
| 125 | Koppelfaktor Achse 4 -> Achse 5 | `float` |
| 126 | Koppelfaktor Achse 4 -> Achse 6 | `float` |
| 127 | Koppelfaktor Achse 5 -> Achse 6 | `float` |
| 128 | Filterfunktion aktiv/passiv | `int` |
| 129 - 140 | Filterwerte für Achsen [ms] | `int` |
| 141 | Filterwert für Bahnen | `int` |
| 142 | Handkonfiguration | `int` |

Definition der Roboterkonstantendatei

Dieses Beispiel macht deutlich, wie schwer das bisherige Konstantenformat zu lesen ist, wenn man nicht die aktuellste Dokumentation vorliegen hat.

Die Konstantendefinitionen werden mit der Weiterentwicklung der Steuerung laufend erweitert. So befinden sich in einer vorliegenden »ROBDAT.SAV«-Datei bereits vier weitere, bisher noch undokumentierte Konstanten.

Dieses verdeutlicht noch einmal den Wunsch nach einem neuen Konstantenformat, das überdies die Möglichkeit des Kommentars bietet.

# 5 XML - eXtensible Markup Language

## 5.1 Kurze Einführung in XML

XML ist eine Abkürzung und steht für eXtensible Markup Language - zu deutsch »Erweiterbare Auszeichungssprache«. Eine Markup-Sprache dient dazu, innerhalb eines Dokumentes, üblicherweise eines Textdokumentes, Teile dieses Dokumentes besonders hervorzuheben, sie auszuzeichnen. Ein sehr bekannter Vertreter dieser Markup-Sprachen ist HTML. HTML ist die Standardsprache zum Erstellen von Web-Seiten. Wenn in einem HTML-Dokument z.B. ein Wort fettgedruckt erscheinen soll, muss dieses Wort entsprechend gekennzeichnet werden. Diese Kennzeichnung umschließt das Wort wie ein Paar Klammern. Diese Kennzeichnungen werden bei HTML wie auch bei XML als Tags bezeichnet. Der Satz

Dies ist ein **fettes** Wort.

sieht in seiner HTML-Entsprechnung wie folgt aus:

```
Dies ist ein <b>fettes</b> Wort.
```

Dabei bezeichnet man `<b>` als öffnendes und `</b>` als schließendes Tag.

Der wesentliche Unterschied zwischen HTML und XML liegt nun darin, dass in HTML ein Satz aller möglicher Tags fest definiert ist. Und jedem dieser Tags ist eine bestimmte Bedeutung zugewiesen. So wird Text, der zwischen `<b>...</b>`-Tags steht, stets fett dargestellt; Text der zwischen `<i>...</i>`-Tags steht, kursiv.

Bei XML hingegen existiert a priori kein definiertes Tag. Die Tags können je nach Bedarf und Anwendungszweck frei definiert werden. Man spricht bei XML deshalb auch von einer Meta-Auszeichnungssprache. Das bietet auf der einen Seite eine nahezu unendliche Freiheit, birgt auf der anderen Seite aber auch die Gefahr der Fehlinterpretation. So wird ein Sportler, der seine Ehrungen verwaltet, ein Tag mit dem Namen `<Preis>` in einem anderen Zusammenhang sehen als ein Einzelhändler, der seine Artikel katalogisieren möchte. Damit ist klar, dass XML zwar aufgrund seiner strengen syntaktischen Regeln von Standardprogrammen gelesen und verarbeitet, nicht aber interpretiert werden kann.

Die vollständige Spezifikation von XML kann unter [5.1] eingesehen werden.

## 5.2 Syntax von XML

Die Bildungsregeln von XML sind einfach und nicht sehr zahlreich.

Ein XML-Dokument beginnt mit einer XML-Deklaration wie folgt

```
<?xml version="1.0" encoding="UTF-8"?>
```

`version` und `encoding` stellen dabei optionale Parameter dar, deren Verwendung allerdings empfohlen wird.

XML-Dokumente sind durch ineinander verschachtelte Elemente aufgebaut. Ein Element besteht immer aus einem öffnenden und einem schließenden Tag und dem Elementinhalt:

```
<Tag>
    Dies ist der Elementinhalt.
</Tag>
```

Ein XML-Element muss immer geschlossen werden. Ein XML-Element kann auch leer sein, es kann dann verkürzt notiert werden: `<LeeresElement />`.

Ein XML-Dokument enthält genau ein XML-Wurzelelement.

XML-Elemente müssen in genau der Ebene geschlossen werden, in der sie geöffnet

```
<Element1>
    Dies ist der Text von Element1
    <Element2>
          Noch mehr Text.
    </Element2>
</Element1>
```

Korrekte Schachtelung von XML-Elementen

```
<Element1>
    Dies ist der Text von Element1
    <Element2>
          Noch mehr Text.
    </Element1>
</Element2>
```

Nicht korrekte Schachtelung von XML-Elementen

wurden:

Die Namen für XML-Tags unterliegen folgender Bildungsregel:

Sie müssen mit einem Unterstrich oder einem Buchstaben beginnen, danach können Buchstaben, Ziffern, Unterstriche, Bindestriche, Punkte oder Doppelpunkte folgen. Die Verwendung von Doppelpunkten ist zwar erlaubt, wird aber nicht empfohlen, da Doppelpunkte eine weitere Sonderfunktion in XML besitzen. Sonderzeichen wie die deutschen Umlaute sind nicht gestattet.

| Gültige XML-Tagnamen | Ungültige XML-Tagnamen |
|---|---|
| <_XMLElement> | <3D-Ansicht> |
| <SPS-Makro> | <-Wert> |
| <Hallo.Du.Da> | <Höhenlage> |
| <Jahr2003> | <Ein Element> |

Beispiele für gültige und ungültige XML-Tagnamen

Ein XML-Dokument, dass diesen Bildungsregeln folgt, wird als »well-formed« oder wohlgeformt bezeichnet.

Darüber hinausgehend wird ein XML-Dokument als gültig (valid) bezeichnet, wenn es nicht nur der XML-Syntax folgt, sondern auch noch die Regeln einer XML-Definitionsdatei einhält.

## 5.3 Definition von XML-Tags

XML erlaubt nicht nur die Bildung von eigenen Tags, sondern erfordert es geradezu. Ohne eigene XML-Tags zu definieren, kann XML nicht sinnvoll verwendet werden. Je nachdem, welche Daten mittels XML gespeichert oder übertragen werden sollen, können andere Tags vonnöten sein. Eine weit verbreitete Methode, Tags zu definieren, ist die sogenannte DTD - die Document Type Definition.

Innerhalb einer DTD wird definiert, welche Tagnamen vorkommen können, welche Elemente ineinander verschachtelt werden dürfen und welche Attribute die einzelnen Tags besitzen. Im folgenden möchte ich auf die DTD-Elemente eingehen, die notwendig sind, um die für die Konstantenspeicherung entworfene DTD zu verstehen:

### 5.3.1 Elemente

Ein Element kann entweder leer sein oder Inhalt besitzen. Beide Fälle werden bei der Definition unterschieden:

Leeres Element: `<!ELEMENT` *Name* `EMPTY>`

Element mit Inhalt: `<!ELEMENT` *Name* `(`*Inhalt*`)>`

Der Elementname muss nach den Bildungsregeln gewählt werden. Der Elementinhalt kann nun aus Text, weiteren Elementen oder beidem bestehen. Es ist aber günstig, wenn man seine Elemente so definieren kann, dass sie entweder Text oder weitere Elemente enthalten. Text wird durch die Zeichenkette `#PCDATA` definiert, weitere Elemente einfach durch ihren Namen. Dabei können die aufgezählten Elemente durch Suffixe in der Häufigkeit ihres Vorkommens festgelegt werden.

Hierzu ein Beispiel:

```
<!ELEMENT Person (Name, Vorname+, Telefon*, Geburtstag?)>
```

Hierbei wird ein Element `Person` definiert, das einige untergeordneten Elemente besitzt. Dabei muss das Element `Name` **genau einmal** vorkommen, das Element `Vorname` **mindestens einmal** (`+`), `Telefon` **beliebig oft** (`*`) - auch keinmal - und `Geburtstag` **kann einmal** (`?`) vorkommen.

Ferner existiert die Möglichkeit, Elemente wahlweise zu verwenden:

```
<!ELEMENT Adresse (Person, (Strasse|Postfach), Ort)>
```

In dieser Definition muss eines der beiden Elemente `Strasse` oder `Postfach` verwendet werden, es können jedoch nicht beide verwendet werden. Der Operator `|` ermöglicht ein einfaches Verodern zweier oder mehr Elemente.

Es ist zu beachten, dass die Reihenfolge der aufgezählten Elemente später innerhalb des XML-Dokumentes einzuhalten ist. Eine Verwendung von

```
<Person>
    <Vorname>Albert</Vorname>
    <Name>Einstein</Name>
</Person>
```

wäre zwar syntaktisch korrekt, entspricht aber nicht der Definition von `Person`, da `Name` vor `Vorname` verwendet werden muss. Welche Möglichkeiten hier zur Erweiterung der Definition gegeben sind, ist in [5.2] nachzulesen.

Weiter ist zu beachten, dass XML-Tags Groß/Kleinschreibung beachten, d.h. `<Person>`, `<person>` und `<PERSON>` stellen drei unterschiedliche XML-Tags dar.

### 5.3.2 Attribute

Attribute sind Name/Werte-Paare, die innerhalb eines XML-Tags verwendet werden können, um dem Tag zusätzliche Informationen hinzuzufügen. Attribute werden immer in Attributlisten definiert. Wenn nur ein Attribut definiert werden soll, besitzt die Attributliste die Länge 1. Eine Attributliste kann, im Gegensatz zu den Elementen, nicht leer sein. Wenn ein Element keine Attribute besitzen soll, wird einfach keine Attributliste definiert. Eine Attributliste besitzt folgenden prinzipiellen Aufbau:

`<!ATTLIST` *Elementname*

*Attributname Typ Verwendung*

...

*Attributname Typ Verwendung*>

Der Elementname bezeichnet das Element, auf welches sich die Attributliste bezieht, d.h. für welches Element die folgenden Attribute definiert werden.

Der Attributname ist, mit den Beschränkungen der XML-Namesbildungsregeln, frei wählbar.

Der Attributtyp bestimmt, in welcher Art der Wert des Attributes interpretiert werden soll. Ingesamt sind in der XML-Version 1.0 zehn verschiedene Attributtypen definiert, von denen hier nur die 4 wichtigsten und häufigsten erläutert werden sollen:

| Typ | Beschreibung |
|---|---|
| `CDATA` | Einfache Zeichenkette |
| `ID` | XML-Name, der innerhalb des ganzen Dokuments eindeutig sein muss, d.h. er darf nicht von einem anderen Attribut des Typs `ID` verwendet werden |
| `IDREF` | Enthält den Wert eines `ID`-Attributes eines Elementes, auf das sich das aktuelle Element bezieht (ID-Referenz) |
| (*Aufzählung*) | Stellt eine Liste von Werten dar, von denen einer als gültiger Attributwert ausgewählt werden kann. |

XML-Attributstypen

Die Verwendung gibt vor, ob das Attribut verwendet werden muss, verwendet werden kann oder einen fest vorgegebenen Wert besitzt:

| Verwendung | Beschreibung |
|---|---|
| `#IMPLIED` | Gibt an, dass das Attribut nicht verwendet werden muss |
| `#REQUIRED` | Fordert, dass das Attribut verwendet werden muss |
| `#FIXED` *Wert* | Gibt einen Wert vor, den das Attribut besitzt. Das Attribut muss nicht explizit angegeben werden. Wird es angegeben, muss es den Wert *Wert* besitzen, andernfalls führt dies zu einem Fehler. |

XML-Attributsverwendungen

Eine Definition und die Verwendung der Attribute im XML-Dokument kann z.B. wie folgt aussehen:

Definition in der DTD:

```
<!ELEMENT Auto EMPTY>
<!ATTLIST Auto
    Kennzeichen ID                    #REQUIRED
    Modell      ("Audi|BMW|Porsche") #REQUIRED
    Farbe       CDATA                 #IMPLIED>
```

Verwendung in dem XML-Dokument:

```
<Auto Kennzeichen="HH-XY 123" Modell="Porsche" Farbe="neon-gelb" />
<Auto Kennzeichen="LG-SE 820" Modell="Audi" />
```

### 5.3.3 Entitäten

Eine Entität, aus dem Lateinischen kommend (ens = das Sein), stellt in XML eine Möglichkeit dar, Referenzen innerhalb einer DTD oder eines XML-Dokumentes zu realisieren. Ähnlich der `#define`-Direktive bei C-Programmen (siehe [5.3]) können so Ersetzungen innerhalb von XML definiert werden.

Man unterscheidet bei XML interne und externe Entitäten. Interne Entitäten werden vollständig in dem XML-Dokument definiert, welches diese Entität verwendet. Externe Entitäten beziehen ihren Inhalt aus externen Quellen, wie einer Datei oder einer URI (a). Externe Entitäten werden häufig verwendet, um einmal erstellte DTD-Dateien in verschiedene XML-Dokumente einzubinden.

Um eine Entität zu verwenden, muss auf sie mittels einer Entitätsreferenz Bezug genommen werden.

In XML-Dokumenten werden in der Regel allgemeine Entitäten referenziert. Eine solche Referenz wird wie folgt referenziert: `&`*Referenz*`;`

In XML existieren 5 vordefinierte Entitäten:

| Entitätsreferenz | Beschreibung |
|---|---|
| `&` | wird zum Zeichen `&` |
| `&apos;` | wird zum Zeichen `'` |
| `>` | wird zum Zeichen > |
| `<` | wird zum Zeichen < |
| `&quot;` | wird zum Zeichen " |

Vordefinierte Entitätsreferenzen in XML

Diese Referenzen können in XML-Dokumenten wie folgt verwendet werden:

```
<Formel>3+6&lt;10</Formel>
```

Innerhalb einer DTD werden sogenannte Parameterentitäten verwendet. Dabei handelt es immer um interne Entitäten innerhalb von DTDen.

Parameterentitätsreferenzen werden mit `%`*Referenz*`;` erstellt. Dabei wird die Referenz textuell durch die Entität ersetzt.

Eine Parameterentität wird wie folgt definiert:

`<!ENTITY %` *Name Inhalt*`>`

Der Name bezeichnet die Entität. Sie wird über diesen Namen referenziert. Der Inhalt der Entität enthält den Text, der bei der Ersetzung eingefügt werden soll.

(a) Eine URI (Universal Resource Identifier) ist eine (weltweit) eindeutige Kennzeichnung für eine ganz bestimmte Datenquelle. Eine URI wird häufig durch eine URL (Universal Resource Locator) beschrieben, die angibt, wo die URI zu finden ist. Häufig verweisen URLs auf bestimmte Stellen im Internet.

Eine Definition einer Parameterentität und deren Verwendung innerhalb einer DTD könnte wie folgt aussehen:

```
<!ENTITY % Farben "rot|gelb|gruen|blau">
<!ELEMENT Auto EMPTY>
<!ATTLIST Auto
    Kennzeichen ID          #REQUIRED
    Farbe       (%Farben;) #IMPLIED>
```

Der Operator `|` stellt hier, wie auch bei den Elementen, eine Veroderung dar. Der Attributwert `Farbe` kann also eine der aufgezählten Farben besitzen: Entweder rot oder gelb oder gruen oder blau.

### 5.4 Kommentare in XML

Eine Forderung, die unter 4.1 an ein neues Datenformat gestellt wird, ist die Möglichkeit, Kommentare zuzufügen. Diese Möglichkeit bietet XML. Kommentare beginnen mit `<!--` und enden mit `-->`. Kommentare können innerhalb eines XML-Dokumentes an jeder Stelle eingefügt werden, mit Ausnahme der allerersten Zeile, da sich dort die XML-Deklaration befinden muss.

### 5.5 Zusammenfassung XML

XML ist eine sehr mächtige und flexible Art, Daten zu strukturieren. Allerdings gibt es auch bei XML einige Nachteile: XML ist zwar hervorragend für Textdaten geeignet, jedoch lassen sich z.B. Binärdaten nur schwer handhaben. Um ein XML-Dokument verarbeiten zu können, muss es ersteinmal überprüft und analysiert werden. Diesen Vorgang bezeichnet man auch als Parsen. Hierzu ist wiederum eine eigene Software, ein Parser, notwendig.

Nichts desto trotz hat XML eine große Verbreitung gefunden - nicht nur im Internet, wo seine ursprüglichen Wurzeln liegen.

Obwohl hier kurz die wesentlichen Grundzüge von XML eingeführt wurden, ist das Thema noch sehr umfangreich. Um sich tiefergehend mit XML und artverwandten Themen zu beschäftigen, wird in [5.2] und [5.4] weitere Literatur zu dem Thema vorgeschlagen. Insbesondere das Werk [5.2] ist für einen einfachen Einstieg geeignet, bietet aber auch dem XML-Erfahrenen noch viel Wissenswertes und eignet sich gut als Nachschlagewerk.

# 6 Das neue Konstantenformat

## 6.1 Anforderungen

Die generellen Anforderungen aus 4.1 an ein neues Datenformat werden allein durch die Wahl von XML erfüllt. Diese sind

- reines Textformat
- Strukturierung der Daten
- Lesbarkeit der Daten
- Einfügen von Kommentaren

Darüber hinaus werden noch eine Reihe weiterer Anforderungen an ein neues Konstantenformat definiert. Diese sind:

- Strukturierung der Konstanten in Kategorien
- Schachtelung der Kategorien in mindestens 3-5 Ebenen
- Schwer zuordbare Konstanten sollen in mehreren Kategorien vorkommen können, dürfen aber nicht redundant gespeichert werden
- Konstanten sollen einzeln in die Steuerung nachladbar sein
- Manuelle Änderungen an der Konstantendatei sollen mit einer gewissen Wahrscheinlichkeit erkennbar sein
- Zusätzlich zu den Konstantenwerten sollen auch die Datentypen festgelegt werden

Alle diese Forderungen können durch eine geeignete Definition einer DTD erfüllt werden. Im folgenden wird die für die Roboterkonstanten erstellte DTD vorgestellt und ihr Aufbau kurz erläutert.

## 6.2 Die Konstantendefinition

Als erstes wird festgelegt, dass alle Namen und Vorgabewerte in Englisch definiert werden. So wird das Sprachproblem, das in 4.1 angesprochen wird, so gut es geht vermieden. Zum anderen ist es damit auch den nicht deutschsprachigen Benutzern der Steuerung möglich, die Konstantendatei zu lesen und zu verstehen.

Auf der folgenden Seite findet sich zunächst einmal die vollständige DTD.

```
<!ENTITY % ItemTypes "int|float|string|date|frac">

<!ELEMENT Root (Categories, Consts, Checksum)>

<!ELEMENT Categories (Category*)>

<!ELEMENT Category (Name, Description?, Category*, ConstRef*)>

<!ELEMENT ConstRef EMPTY>
<!ATTLIST ConstRef
 ref IDREF #REQUIRED
>

<!ELEMENT Consts (Const*)>

<!ELEMENT Const (Name, Description?, Item*)>
<!ATTLIST Const
 id ID #REQUIRED
>

<!ELEMENT Item EMPTY>
<!ATTLIST Item
 name  CDATA          #REQUIRED
 value CDATA          #REQUIRED
 type  (%ItemTypes;) #REQUIRED
>

<!ELEMENT Checksum EMPTY>
<!ATTLIST Checksum
 value CDATA #REQUIRED
>


<!ELEMENT Name (#PCDATA)>

<!ELEMENT Description (#PCDATA)>
```

Definition der Konstanten-DTD

Zu Beginn der DTD wird eine Parameterentität definiert. Hier werden alle Datentypen aufgezählt, die der Steuerung bekannt sein sollen. Es ist empfehlenswert, sämtliche Entitäten zu Beginn einer DTD zu definieren, da sie dann an einer einzigen Stelle stehen und sehr leicht aufzufinden sind.

Danach wird das Wurzelelement definiert. Es benötigt keine weiteren Informationen und dient nur als Container für weitere Elemente. Zur deutlichen Kennzeichnung wird dieses Element mit dem Namen `Root` bezeichnet.

Das Wurzelelement muss nun genau drei untergeordnete Elemente beinhalten: `Categories`, `Consts` und `Checksum`. Die Elemente `Categories` und `Consts` sind ihrerseits nur wieder Container für die einzelnen Kategorien bzw. Konstanten. Man hätte die Konstanten und Kategorien auch direkt unterhalb des Wurzelelements platzieren können. Die feinere Unterteilung dient lediglich der besseren Lesbarkeit und Übersichtlichkeit.

Eine Kategorie besitzt immer einen Namen, danach folgt eine optionale Beschreibung. Falls sie weitere Unterkategorien besitzt, werden diese in der gewünschten Reihenfolge angegeben. Danach folgen, falls vorhanden, Referenzen auf die einzelnen Konstanten. Durch diese Definition können die Kategorien theoretisch beliebig tief verschachtelt werden.

Eine Konstante besitzt immer einen Namen, danach folgt eine optionale Beschreibung. Sie muss eine eindeutige Kennung besitzen, die als ID-Attribut gespeichert wird. Eine Konstante kann beliebig viele Werte enthalten, z.B. Abmaße oder Geschwindigkeiten für bis zu 12 Achsen. Daher folgt nun eine beliebige Anzahl an Werteinträgen, die hier als `Item` definiert sind.

Ein `Item` selbst ist ein leeres Element, d.h. es besitzt weder untergeordnete Elemente noch Elementtext. Die Werteinformationen werden in Form von Attributen gespeichert. Ein `Item` muss genau drei Attribute besitzen: Einen Namen, einen Wert und einen Datentypen. Der Name wird laut Definition in Englisch gehalten, typischerweise dürfte es sich um »`axis[`*n*`]`« handeln, wobei *n* einen zugeordneten Index darstellt. Wie Indizes definiert und ausgewertet werden, wird später genauer erläutert.

Das Attribut `value` enthält den eigentlichen Wert, typischerweise eine Fließkomma- oder Ganzzahl.

Zuletzt folgt ein Attribut, dass den erwarteten Datentyp angibt. Dieser Datentyp muss ein Element der Liste sein, die durch die Entität `ItemTypes` definiert wird.

Als letztes Element unterhalb von `Root` folgt eine Prüfsumme. Dieses Element ist ebenfalls leer und speichert seine Information als Attribut. Die Prüfsumme dient dazu, manuelle Änderungen am XML-Dokument mit einer gewissen Wahrscheinlichkeit zu erkennen. Wie sie berechnet wird, wird später erläutert.

### 6.3 Arbeiten mit der Konstantendefinition

Nachdem die Konstanten-DTD erstellt ist, kann auf ihrer Basis eine XML-Konstantendatei erzeugt werden. Da zunächst noch keine Werkzeuge zum automatisierten Erstellen vorhanden sind, genügt die Kenntnis der DTD, um mittels eines normalen Editors eine solche Datei zu erstellen.

Eine gültige Datei wäre z.B. die folgende:

```
<?xml version="1.0" encoding="UTF-8"?>
<Root>
    <Categories>
          <Category>
                <Name>Reglerkonstanten</Name>
                <Description>Ehemals Datei KONSTDAT.SAV</Description>
                <ConstRef ref="cst _ 100" />
          </Category>
    </Categories>
    <Consts>
          <Const id="cst _ 100">
                <Name>Verstaerkungsfaktor P-Regler</Name>
                <Item name="axis[1]" value="100" type="int"/>
                <Item name="axis[2]" value="100" type="int"/>
                <Item name="axis[3]" value="100" type="int"/>
                <Item name="axis[4]" value="105" type="int"/>
                <Item name="axis[5]" value="100" type="int"/>
                <Item name="axis[6]" value="100" type="int"/>
          </Const>
    </Consts>
    <Checksum value="0" />
</Root>
```

Um korrekte Werte für die Namen-Attribute der `Item`-Elemente zu erstellen, reicht es für XML aus, wenn diese den Bildungsregeln für XML-Namen genügen. Die Forderungen, die im Rahmen der späteren Verwendung gestellt werden, gehen jedoch ein wenig über die normalen XML-Regeln hinaus:

Es wird definiert, dass hier nur englische Ausdrücke verwendet werden sollen.

Wiederkehrende Namen, typischerweise »axis«, werden mittels eckiger Klammern indiziert. Dieses dient dazu, dass im späteren Programmcode der Index extrahiert werden kann und diese Werte über ihren numerischen Index angesprochen werden können. Dieses lässt die Verwendung der C-Anweisung `switch()` zu, während andernfalls eine aufwendige `if() else if`.. Konstruktion mit Zeichenkettenvergleich notwendig wäre.

Die Prüfsumme dient dazu, mit einer gewissen Wahrscheinlichkeit zu erkennen, dass die Datei manuell bearbeitet wurde. In ihrem Attribut `value` wird ein automatisch generierter Wert gespeichert, der einen Zusammenhang mit dem Dateiinhalt aufweist. Diese Prüfsumme manuell zu bilden, ist zwar nicht schwer, aber aufwendig. Der zugrunde liegende Algorithmus arbeitet wie folgt:

Der Text von `<Root>` bis `<Checksum ...` wird auf jedes einzelne Zeichen untersucht. Die (alte) Prüfsumme selbst darf nicht mit in die Berechnung einbezogen werden, da sie das Ergebnis in jedem Falle verfälscht, d.h. unbrauchbar macht.

Der Ordinalwert jedes Zeichens abzüglich 32 wird summiert. Die Gesamtsumme, als hexadezimale Zahl formatiert, ergibt dann die Prüfsumme. Dieses Verfahren ist einfach und dennoch einigermaßen wirkungsvoll. Änderungen, wie Hinzufügen oder Löschen werden sicher erkannt. Modifikationen von Zeichen werden erkannt, wenn nicht zufällig (oder absichtlich) an anderer Stelle Zeichen zugefügt, gelöscht oder so verändert werden, dass sich dieselbe Gesamtsumme ergibt. Das Verfahren bietet einen recht wirkungsvollen Schutz gegen Datenverlusten auf Datenträgern oder versehentlich veränderten Dateien. Es wird hierbei nicht der Anspruch der absoluten Manipulationssicherheit erhoben.

Der Wert 32 wird subtrahiert, da 32 der Ordinalwert des Leerzeichens ist. Jedes Leerzeichen wird somit mit dem Wert Null addiert, so dass es keine Auswirkungen auf die Prüfsumme besitzt. Dieses ist so gewählt, um einerseits die Prüfsumme nicht zu groß werden zu lassen, andererseits auch, um die Prüfsumme von der Formatierung des Textes ein wenig unabhängiger zu gestalten. Steuerzeichen, deren Ordinalwerte kleiner als 32 sind, werden somit von der Prüfsumme subtrahiert. Jedoch ist ihre Anzahl im Vergleich mit der Menge der vorkommenden Zeichen sehr gering, so dass die Prüfsumme dennoch keinen negativen Wert annehmen dürfte. Falls dies jedoch aus beliebigen Gründen einmal eintreten sollte, ist die Funktion dennoch wirkungsvoll, denn eine negative Prüfsumme ist sowohl zulässig als auch reproduzierbar.

Der C++ Code, der diese Berechnung erledigt, gestaltet sich denkbar einfach:

```
for (int i=0; i<XML.Length(); i++)
  Checksum += ((int) XML[i+1]) - 32;
```

Dabei ist `XML` eine Instanz der Klasse `AnsiString`, wie sie z.B. in dem Borland C++Builder implementiert ist. Über die Methode `.Length()` wird hierbei die Länge des Strings ermittelt, der Index `XML[i+1]` greift auf die einzelnen Zeichen des Strings zu, genau wie bei einer klassischen `char*` - Definition.

Obgleich der Algorithmus einfach ist, so ist er bei langen Dateien manuell fast nicht mehr ausführbar. Aus diesem Grund, und um eine komfortable Umgebung zur Erzeugung von Konstantendateien zu schaffen, wurde der XML-Konstanteneditor geschrieben.

# 7 Der Konstanteneditor

## 7.1 Die Grundidee

Die Idee, die hinter der Entwicklung des Konstanteneditors stand, war es, eine komfortable und übersichtliche Umgebung zu schaffen, mittels derer Konstantendateien bearbeitet werden können. Dabei sollen die Möglichkeiten, die sich aus der Definition der DTD ergeben, wie z.B. die Schachtelung der Kategorien, so implementiert werden, dass sie einfach genutzt werden können.

Zudem soll der Editor eine Reihe von automatisierten Prüfungen übernehmen:

- Bilden und Verifizieren der Prüfsumme
- Sicherstellen, dass keine Konstanten-IDs mehrfach vergeben werden
- Überprüfen, ob Wert und Typ eines Konstanten-Eintrags übereinstimmen

Der Editor braucht nicht zu überprüfen, welche Konstanten von einem Benutzer geändert werden dürfen, da er sie im Nachhinein innerhalb der XML-Datei ändern könnte. Diese Sicherheitsüberprüfung ist in der Steuerung selbst implementiert.

Darüber hinaus soll es möglich sein, Teile einer (Master-)Konstantendatei zu exportieren, um somit Konstanten-Untermengen zu bilden. Dieses Verfahren kann dazu benutzt werden, einzelne Konstanten in die Steuerung nachladen zu können.

## 7.2 Die Implementation

Der Konstanten-Editor wurde unter Microsoft Windows XP mit dem Borland C++ Builder 6 entwickelt. Diese Entwicklungsumgebung wurde gewählt, da C/C++ die im Hause der SEF Roboter GmbH übliche Programmiersprache ist. Zudem ist in dem C++Builder 6 bereits ein einfacher XML-Parser integriert auf dem die Entwicklung aufsetzen kann. Weiterführende Literatur zu der Programmierung mit dem C++Builder findet sich unter [7.1] und [7.2].

## 7.3 Die Bedienung

### 7.3.1 GRUNDLAGEN

Um die Bedienung und den Funktionsumfang des Konstanten-Editors zu erklären, ist dieses Kapitel als Anleitung aufgebaut, in der schrittweise eine leere Konstantendatei gefüllt wird, bis sie Kategorien, Unterkategorien, Konstanten und deren Werte enthält. Dabei wird auf den Unterschied zwischen Konstanten und Konstantenreferenzen eingegangen und schließlich wird der Datenexport erläutert.

Der Editor kann mehrere Konstantendateien gleichzeitig geöffnet haben. Jede Datei wird in einem eigenen Reiter im Hauptfenster angezeigt. Grundsätzlich kann jedes Fenster des Editors mit `[ESC]` geschlossen werden. Popup-Menüs können Windows-typisch mit der rechten Maustaste eingeblendet werden.

Die kleinste, gültige Konstantendatei (entsprechend der DTD) ist wie folgt aufgebaut:

```
<?xml version="1.0" encoding="UTF-8"?>
<Root>
    <Categories>
    </Categories>
    <Consts>
    </Consts>
    <Checksum value="0" />
</Root>
```

Diese Datei soll als Basis für dieses Kapitel dienen. Sie kann schnell manuell erstellt oder aus einer bestehenden Datei erzeugt werden. Da die Prüfsumme nicht bekannt ist, wird sie einfach auf Null gesetzt.

Wenn diese Datei nun in den Editor geladen wird, erscheinen zwei Hinweise. Zunächst wird auf den Prüfsummenfehler hingewiesen. Wenn die Datei geladen wird, wird automatisch die gültige Prüfsumme ermittelt. Um die korrekte Prüfsumme in die Datei zu schreiben, muss diese nur gespeichert werden. Der Hinweis »DATEI GEÄNDERT« verschwindet aus der Statusleiste des Hauptfensters.

Danach erscheint der Hinweis, dass keine Kategorien gefunden wurden. Eine Konstantendatei sollte immer mindestens eine Kategorie besitzen. Konstanten werden nur dann angezeigt, wenn sie mindestens einer Kategorie zugeordnet sind.

### 7.3.2 KATEGORIEN UND UNTERKATEGORIEN

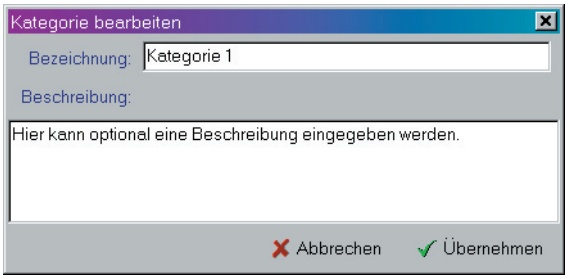
Aus diesem Grund müssen zunächst die Kategorien angelegt werden. Dieses erfolgt über das Popup-Menü der Kategorien-Anzeige:


Konstanteneditor: Neue Hauptkategorie anlegen

Eine Hauptkategorie ist eine Kategorie, die sich direkt unterhalb des `Categories`-Elements befindet. Eine neue Hauptkategorie wird immer oben in der Liste eingefügt. Es ist jedoch möglich, diese Reihenfolge später wieder zu ändern.

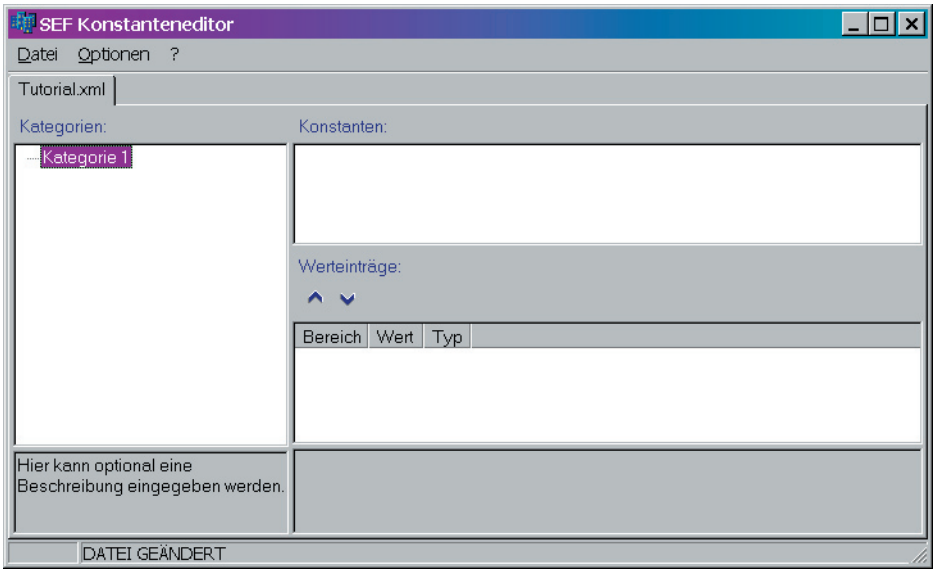
Die Verschachtelung und Reihenfolge der Kategorien spielt bei der Auswertung der Datei auf der Steuerung keine Rolle. Die Kategorien dienen lediglich der übersichtlicheren Verwaltung der Konstanten mit Hilfe des Konstanteneditors.

Nachdem der Menüpunkt zur Erstellung einer neuen Hauptkategorie ausgewählt wurde, öffnet sich ein Fenster, in dem die Daten der Kategorie eingegeben werden können. Dasselbe Fenster wird auch verwendet, wenn Daten einer vorhandenen Kategorie bearbeitet werden.




Konstanteneditor: Kategorie bearbeiten

Nachdem die Texte übernommen wurden, erscheint die neue Kategorie in der Anzeige:

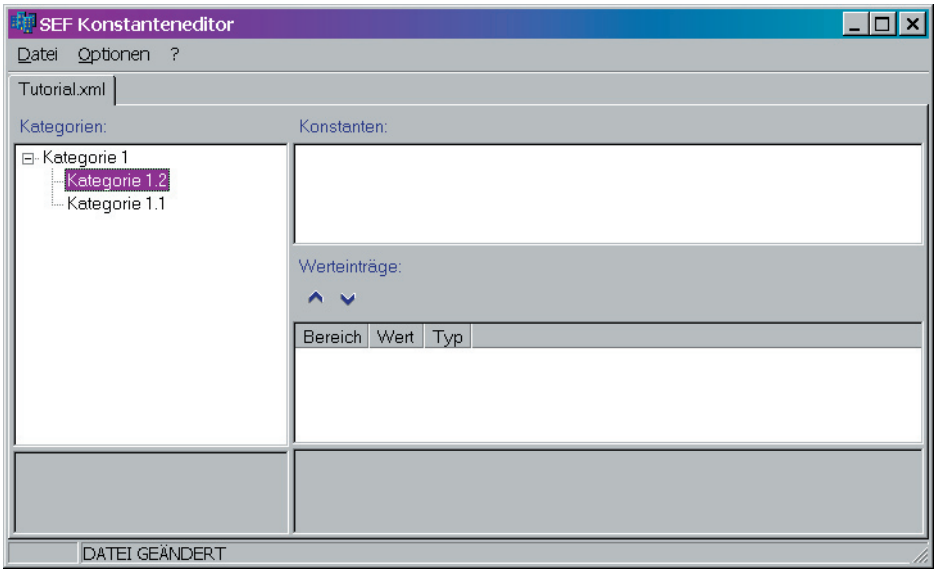



Konstanteneditor: Hauptfenster mit einer Hauptkategorie

Auf ähnliche Weise werden nun zwei Unterkategorien angelegt. Dazu muss die Kategorie, die eine Unterkategorie erhalten soll, ausgewählt sein. Über das Popup-Menü wird jetzt der Punkt »Neue Kategorie« gewählt. Es erscheint wieder das Fenster zur Eingabe der Kategorie-Daten. Auf diese Weise können unter »Kategorie 1« zwei Unterkategorien angelegt werden: »Kategorie 1.1« und »Kategorie 1.2«.

Dabei ist zu beachten, dass auch hier die neu zugefügte Kategorie immer als erste Unterkategorie eingefügt wird.

Nachdem die beiden Unterkategorien erstellt wurden, erscheint dies also wie folgt im Konstanteneditor:



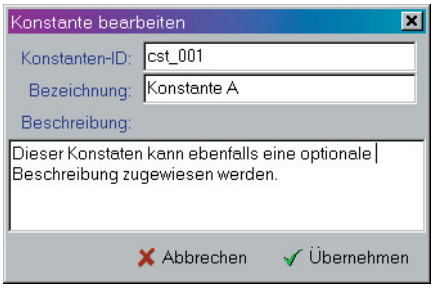

Konstanteneditor: Hauptfenster mit zwei Unterkategorien

### 7.3.3 KONSTANTEN

Nachdem nun Kategorien vorhanden sind, können diesen Konstanten zugeordnet werden. Zunächst einmal müssen Konstanten erstellt werden. Dieses ist über das Konstanten-Popup-Menü möglich:


Konstanteneditor: Popup-Menü Konstanten

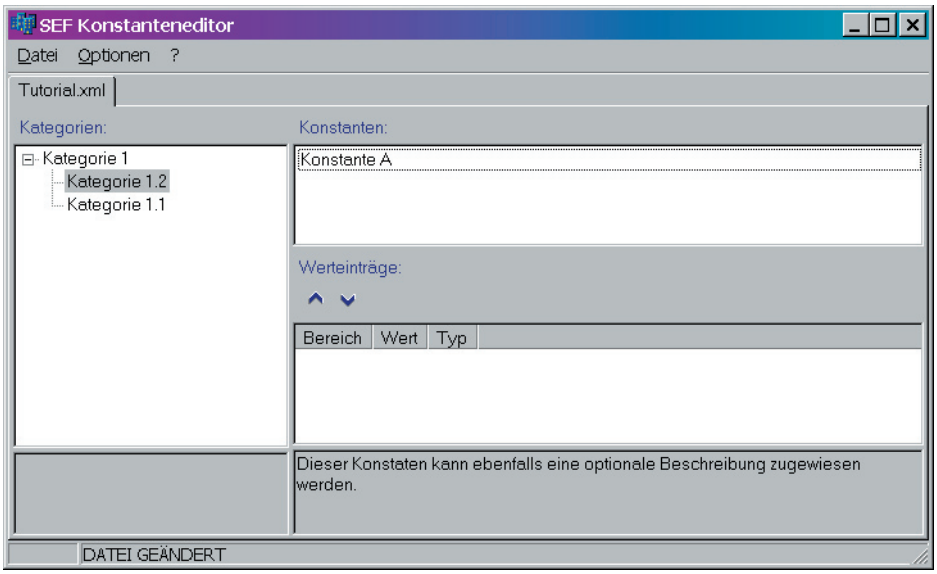
Nun öffnet sich ein Fenster, in dem die Konstante beschrieben wird. Dabei ist darauf zu achten, dass die Konstanten-ID den XML-Bildungsregeln genügt, da sie später als Attributwert gespeichert wird. Sie darf also nicht mit einer Ziffer beginnen.




Konstanteneditor: Konstante bearbeiten

Falls eine Konstanten-ID doppelt vergeben wurde, erscheint beim Übernehmen eine entsprechende Warnung. Die Daten können in diesem Fall nicht übernommen werden. Ebenso ist es nicht möglich, keine ID zu vergeben.

Nachdem die Konstante übernommen wurde, erscheint sie auch in dem Hauptfenster, wenn die Kategorie ausgewählt wird, der sie zugeordnet ist:




Konstanteneditor: Hauptfenster mit Konstante

Danach wird eine weitere Konstante mit dem Namen »Konstante B« erzeugt.

### 7.3.4 KONSTANTENREFERENZEN

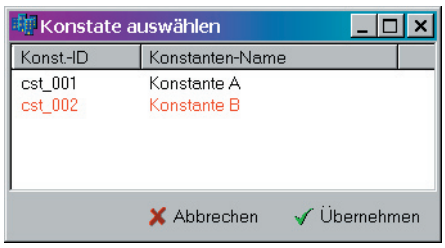
Wenn eine Konstante neu angelegt wird, dann werden automatisch zwei Elemente erzeugt: Zum einen die Konstante an sich, die mit all ihren Eigenschaften unter dem Element `Consts` in der XML-Datei gespeichert wird. Gleichzeitig wird eine Konstantenreferenz erstellt, die unter der gerade ausgewählten Kategorie eingefügt wird.

Es ist nun möglich, diese Referenzen zu entfernen oder einer Kategorie neue Referenzen hinzuzufügen, ohne eine Konstante zu löschen oder neu anzulegen. Eine Referenz kann selbst dann entfernt werden, wenn es die letzte Referenz auf diese Konstante ist. Es soll nun die Referenz auf die gerade erstellte »Konstante B« gelöscht werden. Dieses ist wieder über das Konstanten-Popup-Menü möglich.

Es ist jedoch wichtig zu wissen, dass eine Konstante auch dann von der Steuerung eingelesen wird, wenn sie nicht referenziert wird. Um ein Einlesen zu verhindern, muss die Konstante an sich aus der Datei entfernt werden.

Einer Kategorie kann eine Konstantenreferenz über das Konstanten-Popup-Menü zugewiesen werden. Es erscheint ein Auswahlfenster, bei der alle Konstanten, die nicht bereits in mindestens einer Kategorie referenziert werden, rot markiert sind.

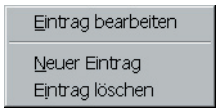
Im vorliegenden Fall handelt es sich um die »Konstante B«, deren letzte Referenz im vorigen Schritt entfernt wurde.




Konstanteneditor: Konstantenreferenz auswählen

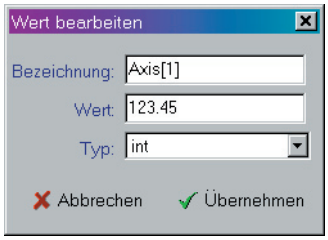
### 7.3.5 KONSTANTENWERTE

Nachdem die Konstanten angelegt sind, können ihnen Werteinträge zugefügt werden. Dieses erfolgt ebenfalls über ein Popup-Menü.




Konstanteneditor: Popup-Menü Konstantenwerte

Wie gewohnt lässt sich der Konstantenwert in einem weiteren Fenster bearbeiten. Hier werden Name, Wert und Typ angegeben. Für den Namen, der hier gewählt wird, gelten die unter 6.3 definierten Bedingungen.

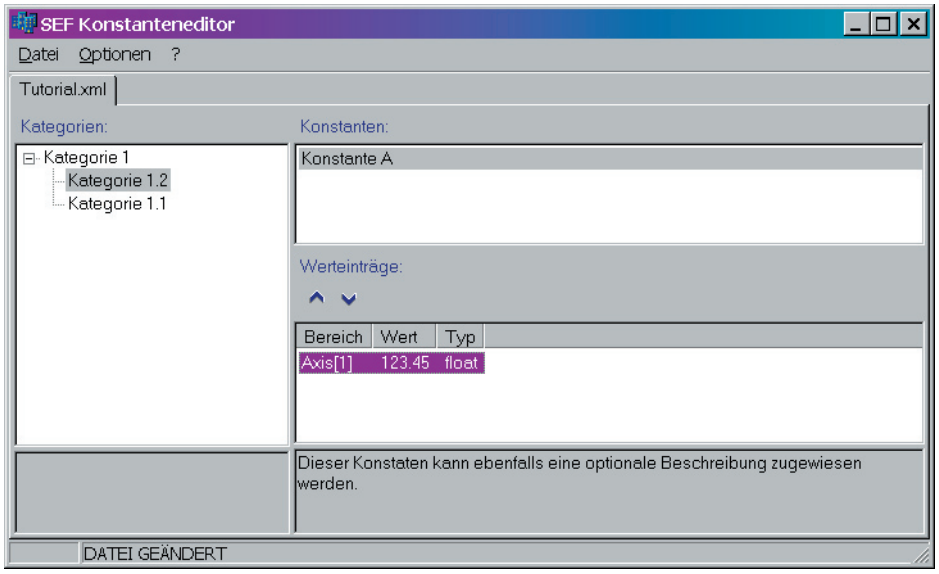



Konstanteneditor: Konstantendefinition mit falscher Typdefinition

Es ist darauf zu achten, dass der ausgewählte Typ und der Wert übereinstimmen. Sollte dies beim Übernehmen nicht der Fall sein, können die Daten nicht eingetragen werden.


Konstanteneditor: Nicht passender Konstantenwert und -typ

Nachdem der Wert mit korrekter Typdefinition übernommen wurde, erscheint er als Element der Konstanten im Hauptfenster des Editors:




Konstanteneditor: Hauptfenster mit Konstantenwerten

Für eine Konstante können nun beliebig viele Werte definiert werden. Theoretisch ist es möglich, den einzelnen Werteinträgen unterschiedliche Typen zuzuordnen. In der Praxis ist dieses jedoch nicht sinnvoll, weshalb darauf geachtet werden sollte, gleiche Typen innerhalb einer Konstanten zu verwenden.

Mit Hilfe des Editors ist es nun möglich, Kategorien, Unterkategorien, Konstanten und deren Referenzen und Werte zu erzeugen, zu bearbeiten und zu löschen.

### 7.3.6 Reihenfolge verändern

Die Strukturierung der Konstanten in Kategorien und Unterkategorien ist ein Hilfsmittel, um dem Bediener den Umgang mit den Konstanten zu vereinfachen. Es soll ihm helfen, Konstanten leichter aufzufinden. Für die Steuerung ist diese Struktur nicht von Bedeutung.

Es ist möglich, die Anordnung der Einträge im Nachhinein zu verändern. Dieses ist sehr einfach über die Tastatur zu erledigen. Ein Eintrag, sei es nun eine Kategorie, eine Konstante oder ein Konstantenwert, können in Ihrer Position vertikal verschoben werden, indem sie zunächst markiert werden. Danach muss die Tastenkombination `[Ctrl]+[↑]` gedrückt werden, um diesen Eintrag nach oben, `[Ctrl]+[↓]` um den Eintrag nach unten zu verschieben.


Konstanteneditor: Verschieben von Einträgen

Kategorien werden mitsamt ihrer Unterkategorien verschoben. Dabei können sie nur in ihrer aktuellen Ebene verschoben werden. Die Ebene einer Kategorie kann mit den Tastenkombinationen `[Ctrl]+[→]` und `[Ctrl]+[←]` geändert werden. Um eine Kategorie zu einer Unterkategorie zu machen, muss sie sich

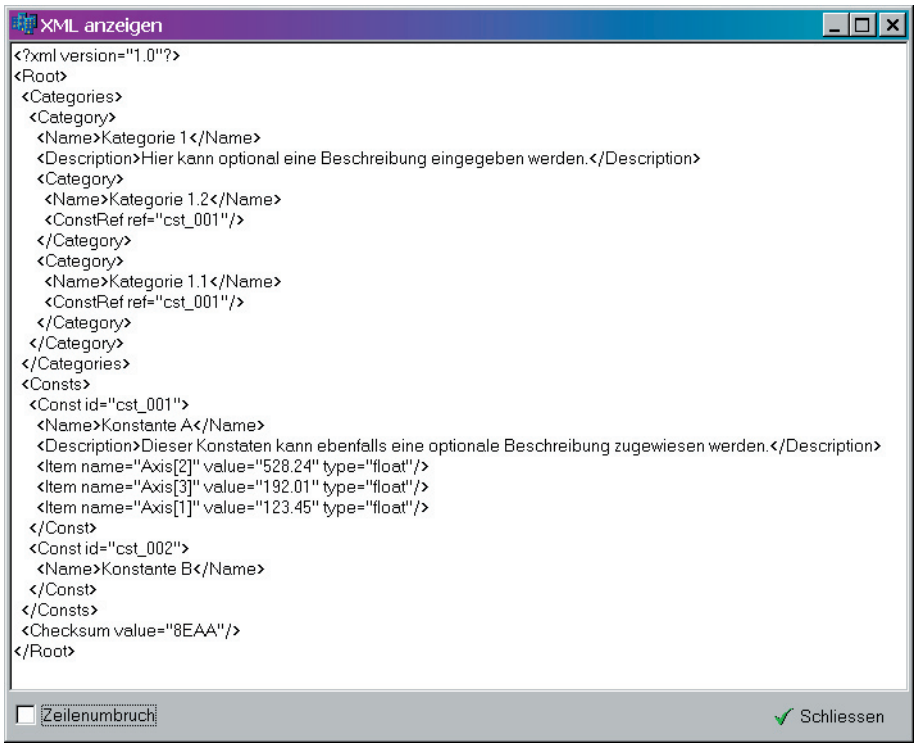
direkt über der Kategorie befinden, in die sie verschoben werden soll und sie muss markiert sein. Durch Drücken von [Ctrl]+[→] wird die markierte Kategorie nun die erste Unterkategorie der darunterstehenden Kategorie. Mögliche vorhandene Unterkategorien der verschobenen Kategorie werden ebenfalls mit verschoben.


Konstanteneditor: Verändern von Kategorieebenen

Durch Drücken von [Ctrl]+[←] wird eine Kategorie samt ihrer Unterkategorien eine Ebene nach oben verschoben. Dabei wird sie genau über der Kategorie eingefügt, der sie zuvor untergeordnet war.

### 7.3.7 XML-TEXT ANZEIGEN

Der Editor ermöglicht den einfachen und übersichtlichen Umgang mit den Konstantendateien. Es kann jedoch auch der entsprechende XML-Text der Datei angezeigt werden. Dies erfolgt über das Hauptmenü »Optionen->XML anzeigen«. Dabei wird derjenige XML-Text angezeigt, der die aktuelle Struktur des Editors abbildet. Dieser kann von dem Text der originalen Konstantendatei abweichen, falls diese nach dem Laden bearbeitet wurde. In diesem Fall wird in der Statusleiste der Hinweis »DATEI GEÄNDERT« angezeigt.




Konstanteneditor: XML-Text anzeigen

### 7.3.8 DATEN EXPORTIEREN

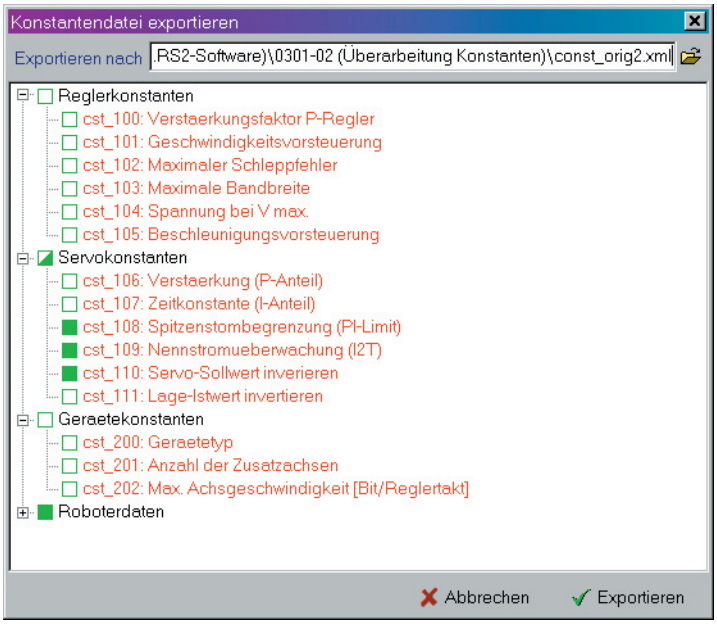
Um eine Teilmenge der aktuellen Konstanten zu erstellen, können diese exportiert werden. Diese Funktion ist dann nützlich, wenn aus einer Masterkonstanten-Datei einzelne Konstanten herausgezogen werden sollen, um diese in der Steuerung zu aktualisieren.

Viele Konstanten dienen dazu, die Applikation, in die der Roboter eingebunden ist, zu optimieren. Sie besitzen nur für diese Applikation eine Gültigkeit. Oftmals werden die optimalen Konstantenwerte nur langsam, empirisch ermittelt.

Falls nun werkseitig eine Konstante geändert werden muss, überschreibt das bisherige Konstantenformat ebenfalls alle anderen Werte. Mit dem neuen Format ist es nun möglich, Dateien mit einer oder wenigen Konstanten zu erzeugen, die dann in die Steuerung nachgeladen werden können, ohne dass die anderen Werte zurückgesetzt werden.

Um Daten zu exportieren, wird über das Hauptmenü »Datei->Exportieren« angewählt. Es erscheint ein Fenster mit einer Baumansicht aller vorhandenen Kategorien und den darin enthaltenen Konstanten. Dabei werden Kategorien in schwarz, Konstanten in rot dargestellt.

Um eine oder mehrere Daten auszuwählen, klickt man auf das grüne Kästchen vor dem Eintrag. Ist es ausgefüllt, wird dieser Eintrag, einschließlich aller Untereinträge, zum Export ausgewählt. Ein halb gefülltes Kästchen weist darauf hin, dass nur einige Untereinträge ausgewählt sind. Untereinträge werden automatisch an- oder abgewählt, falls ein übergeordnetes Element explizit markiert oder die Markierung entfernt wird.




Konstanteneditor: Daten exportieren

# 8 XML auf der S.RS2

## 8.1 XML-Parser

### 8.1.1 Allgemeines

XML-Parser sind Softwarepakete, die die Funktionalität bereitstellen, XML-Dateien zu analysieren. Dazu werden die Dateien eingelesen und auf ihre syntaktische Korrektheit überprüft.

Zusätzlich können einige Parser überprüfen, ob die gelesenen XML-Dateien den Regeln entsprechen, die in der DTD für diese Datei angegeben sind. Diese Parser werden als validierende Parser bezeichnet.

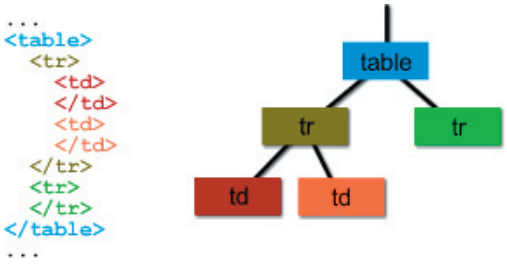
Bei XML selbst handelt es sich nicht um eine Software, sondern um einen offenen Standard. Das bedeutet, dass zwar Definitionen, Bildungsregeln und erwartete Funktionalitäten offen dargelegt, diese aber nicht implementiert werden. Es steht jedem frei, eine XML-konforme Software, z.B. einen Parser, zu entwickeln. Aus diesem Grund existieren eine Vielzahl verschiedener XML-Parser, die miteinander konkurrieren. Die meisten von Ihnen sind freie Software. Viele freie Parser sind zudem im Sourcecode verfügbar und lassen sich somit in eigene Projekte integrieren oder auch auf fremde Zielplattformen portieren.

Bei der Aufgabe, einen XML-Parser in die Steuerung S.RS2 zu implementieren, muss zunächst entschieden werden, ob der Parser selbst geschrieben werden soll oder ob ein existenter Parser verwendet werden kann. Falls ein vorhandener Parser verwendet werden soll, muss dieser in das Betriebssystem VxWorks integrierbar sein oder als Source vorliegen. Zudem sollte er gut dokumentiert und schon erprobt sein. Außerdem sollte er nicht zu umfangreich sein und auch nach der Kompilation nicht zuviele Systemresourcen belegen.

Nachdem der Parser das XML-Dokument untersucht hat, muss er dafür sorgen, dass die Informationen dem eigenen Programm zur Verfügung gestellt werden. Wenn man sich die vorhandenen Parser betrachtet, lässt sich feststellen, dass diese in zwei Kategorien eingeteilt werden können: DOM-Parser und SAX-Parser.

### 8.1.2 DOM-PARSER

Eine Art, die Information des XML-Dokuments bereitzustellen, ist es, die Struktur des Dokuments im Speicher des Rechners abzubilden. Diese Struktur wird als DOM bezeichnet, was für »Document Object Model« steht. Dabei werden die Elemente des Dokuments als Knoten in einen Objektbaum eingefügt.




DOM-Repräsentation einer Tabelle

Dabei stellt der DOM-Parser eine Referenz auf den Wurzelknoten des Baums zur Verfügung, welcher das Wurzelelement des XML-Dokuments repräsentiert. Von dort aus kann durch den gesamten Baum, der vollständig im Speicher vorhanden ist, navigiert werden. Die Knoten werden dann über ihre relativen Beziehungen zueinander bestimmt wie Eltern-, Kind- und Schwesterelemente. Meistens sind ebensfalls Funktionen implementiert, die einzelne Knoten über Bedingungen finden, z.B. deren Namen oder ein `ID`-Attribut. Diese Art, auf Knoten zuzugreifen, wird als XPath bezeichnet. Die genaue Definition von XPath findet sich unter [8.1].

Der Vorteil der DOM-Darstellung liegt darin, dass das Dokument nur einmal gelesen werden muss. Danach ist es für den weiteren Gebrauch im Speicher abgelegt. Dieses stellt bei großen XML-Dokumenten allerdings hohe Anforderungen an den benötigten Arbeitsspeicher. Insbesondere bei Client/Server-Anwendungen bei denen serverseitig mit XML-Dokumenten gearbeitet wird, ist diese Darstellung ungeeignet, wenn für jede Client-Anfrage ein eigenes DOM aufgebaut werden muss, wie es z.B. bei Web-Server-Anwendungen der Fall ist.

Die Spezifikation, wie ein DOM aufgebaut sein muss, ist vom W3C definiert. Sie ist, wie alle vom W3C veröffentlichen Definitionen, auf ihrer Homepage nachzulesen. Die Definition des DOM findet sich unter [8.2].

### 8.1.3 SAX-PARSER

Eine völlig anderer Ansatz, die XML-Daten zur Verfügung zu stellen, ist die stream- oder ereignisorientierte Methode. Hierbei wird die XML-Datei sequentiell ausgewertet, d.h. sie wird eingelesen und jedesmal, wenn der Parser einen Knoten entdeckt, wird durch den Parser eine Callback-Funktion aufgerufen. Diese Funktion wird vor Beginn des Parsens bei dem Parser angemeldet.

Die XML-Datei kann in Puffern beliebiger Größe geparst werden. Somit ist der Speicherplatzbedarf von der Größe der Datei unabhängig und in etwa vorhersagbar.

Eine Parser-Implementation, die diese ereignisorientierte Methode verwendet, ist SAX. SAX ist eine Abkürzung und steht für »Simple API for XML«. Der Parser SAX, geschrieben unter der Leitung von David Megginson[8.3], wurde speziell für Client/Server-Applikationen entwickelt. SAX ist klein, schnell und kann somit eine Reihe von Clientanfragen gleichzeitig beantworten.

In Anlehnung an den Erfolg von SAX setzte sich dieser Name als allgemeine Bezeichnung für ereignisorientierte Parser durch.

Das folgende Beispiel soll erläutern, wie SAX-Parser funktionieren. Links ist die XML-Datei dargestellt, rechts die Liste der vom Parser erzeugten Callback-Aufrufe.

```
<?xml version="1.0"?>
<Root>
  <Person id="p1">
    <Vorname>George</Vorname>
    <Name>Bush</Name>
  </Person>
  <Person id="p2">
    <Vorname>Arnold</Vorname>
    <Name>Schwarzenegger</Name>
  </Person>
  …
</Root>
```

```
startDocument
startElement   (Root)
startElement   (Person p1)
startElement   (Vorname)
characters     (George)
endElement     (Vorname)
startElement   (Name)
characters     (Bush)
endElement     (Name)
endElement     (Person)
startElement   (Person p2)
startElement   (Vorname)
characters     (Arnold)
endElement     (Vorname)
startElement   (Name)
characters     (Schwarzenegger)
endElement     (Name)
endElement     (Person)
…
endElemet      (Root)
endDocument
```

Darstellung der Funktionsweise eines SAX-Parsers

## 8.2 Auswahl eines Parsers

### 8.2.1 XERCES

Einer der bekanntesten und am weitesten verbreiteten Parser ist Xerces von Apache [8.4]. Xerces ist in drei Implementation für C++, Java und Perl verfügbar. Der Funktionsumfang von Xerces ist sehr groß. Xerces ist ein validierender Parser, der viele W3C-Standards implementiert hat:

- XML 1.0 und 1.1
- DOM Level 1 und Level 2
- SAX 1.0 und 2.0
- XML Schema
- Namespaces in XML

In der folgenden Tabelle sind die drei größten Vor- bzw. Nachteile, Xerces auf der S.RS2 zu verwenden, aufgezählt:

| Vorteile | Nachteile |
|---|---|
| Parser im Source verfügbar | Source ist in C++ geschrieben, nicht in C |
| Parser unterstützt nahezu jeden W3C-Standard | Das Kompilat ist sehr groß, nur wenige Funktionen werden wirklich benötigt |
| Parser ist erprobt und sehr stabil | Portierung auf VxWorks ist langwierig und umständlich |

Vor- und Nachteile des Einsatzes von Xerces auf der S.RS 2

Um zu entscheiden, ob Xerces für den Einsatz auf der S.RS2 geeignet erscheint, betrachtet man am besten die Anforderungen an den Parser. Er sollte möglichst als C-Source vorliegen, da C++ auf einem Realtime-System nur bedingt verwendet werden sollte. C++ ermöglicht sehr komplexe Programmstrukturen, insbesondere im Zusammenhang mit objektorientierter Programmierung, wodurch zusätzliche Überprüfungen von Laufzeiten notwendig werden können.

Zudem ist die Funktionalität, die Xerces bietet, deutlich überdimensioniert gegenüber dem, was auf der Steuerung benötigt wird. Hier reicht es, XML lesen zu können und die erhaltenen Daten in die internen Strukturen der Steuerung zu schreiben.Operation auf den XML-Daten, die bedingen, dass die Daten im Speicher gehalten werden müssen, sind derzeit nicht notwendig.

Aus diesen Gründen fällt Xerces, obwohl weit verbreitet und erprobt, zunächst nicht in die engere Wahl.

### 8.2.2 eXpat

Ein anderer Parser, der in Betracht kommt, ist eXpat von James Clark [8.5]. James Clark war einer der führenden Köpfe bei der Implementation der ersten XML-Spezifikationen. Seine Parser, unter anderem auch sein Java-orientierter Parser XP, sind nach Xerces am weitesten verbreitet und bilden den Kern vieler professioneller Anwendungen. So wird z.B. eXpat im Netscape Navigator verwendet.

eXpat ist in ANSI-C geschrieben und steht als Sourcecode zur Verfügung. Es handelt sich um einen freien Parser. Er darf ohne Beschränkungen kopiert, in eigene Projekte eingebunden, modifiziert oder verkauft werden.

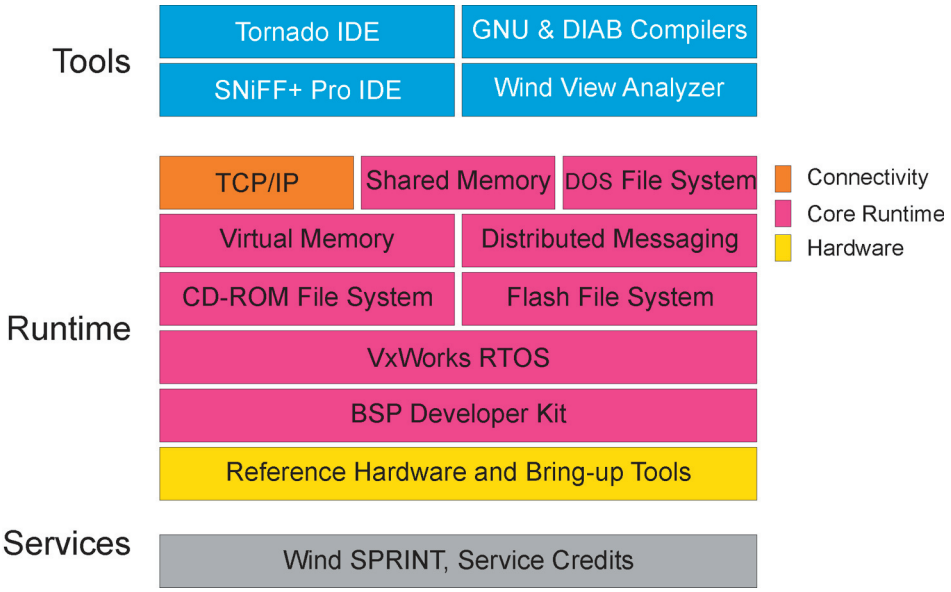
eXpat ist ein SAX-Parser. Er besteht aus wenigen Quell-Dateien und ist auch im Kompilat nur ca. 350 kB groß. Für eXpat existiert eine gute Dokumentation und viele News-Foren im Internet beschäftigen sich mit diesem Parser.

Eine Portierung auf VxWorks ist mit nur geringem Aufwand möglich. Aus diesen Gründen wurde eXpat gewählt, um die XML-Schnittstelle auf der S.RS2 zu realisieren.

## 8.3 Implementierung der Schnittstelle

### 8.3.1 Die Entwicklungsumgebung

Um eine Software für das Betriebssystem VxWorks zu entwickeln, bietet WindRiver eine Reihe von Tools an, unter anderem eine integrierte Entwicklungsumgebung mit dem Namen Tornado, die für MS Windows und Sun Solaris zur Verfügung steht.




Basis-Entwicklungstools für VxWorks

Die Grafik verdeutlicht, dass WindRiver von einer Referenzhardware, über den Realtime-Betriebssystemkern und verschiedene Module bis hin zu Entwicklungs- und Testumgebungen eine vollständige Real-Time-Lösung anbietet.

Eine integrierte Entwicklungsumgebung (engl. integrated development environment, kurz IDE) vereint unter einer einheitlichen Oberfläche alle Tools, die notwendig sind, ein Programm zu erstellen. Dazu zählen insbesondere der Editor, Compiler, Linker und Debugger sowie weitere Test- und Organisationstools wie z.B. ein Projektmanager.

Obwohl VxWorks wie auch Tornado kommerzielle Produkte der oberen Preisklasse sind, sind sie dennoch nicht frei von Fehlern. Viele kleine, meist auf den ersten Blick nicht erkannbare Fehler und Instabilitäten erschweren das Arbeiten mit diesen Produkten. Zudem ist die Onlinehilfe mangelhaft und unvollständig. Es existiert jedoch eine umfangreiche schriftliche Dokumentation in englischer Sprache, die die einzelnen Werkzeuge beschreibt [8.6].

Wind River bietet darüber hinaus die Simulationssoftware VxSim an, mit deren Hilfe der geschriebene Quelltext getestet werden kann, ohne ihn auf eine Zielplattform herunterladen zu müssen. Leider ist die Version, die mit Tornado ausgeliefert wird, nur eingeschränkt funktionsfähig. Bereits die C-Standardfunktion `fopen()` zum Öffnen einer Datei wird nicht unterstützt.

Da diese Funktion jedoch für das Einlesen der XML-Konstantendateien unabdingbar ist, kann der Quellcode nur auf dem Zielsystem vollständig getestet werden.

### 8.3.2 Die Zielplattform

Tornado ist in der Lage, Quellcode mit unterschiedlichen Compilern zu verarbeiten, so dass der Code für unterschiedliche Zielsysteme erstellt werden kann. In diesem Fall ist die Zielplattform ein Intel Pentium Prozessor.

Rechnereinheit der S.RS2

Die Hardware, auf der die Software laufen soll, ist die Rechnereinheit der Robotersteuerung S.RS2. Die anderen Komponenten der Steuerung, wie Servomodul oder PHG, werden dazu nicht benötigt.

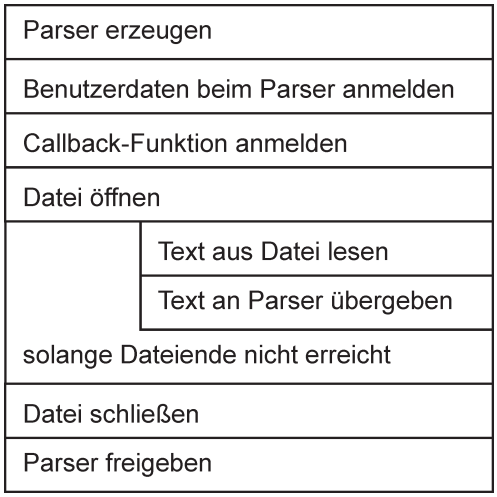
Zunächst muss die Rechnereinheit über die serielle Schnittstelle mit dem Entwicklungsplatz verbunden werden. Danach wird das Zielsystem gestartet und VxWorks geladen. Nun kann der erzeugte Code von der Entwicklungsumgebung in die Steuerung heruntergeladen werden. Tornado stellt eine Shell zur Verfügung, die sich ebenfalls mit dem Zielsystem verbindet. Über diese Shell kann das Zielsystem überprüft und gesteuert werden. So ist es möglich, einzelne C-Funktionen des heruntergeladenen Codes aufzurufen, z.B. die selbstgeschriebene Funktion `testit()` aus dem Projekt der XML-Implementation.

### 8.3.3 DAS PROGRAMM

Nachdem der Parser eXpat so angepasst wurde, dass er unter Tornado compiliert werden konnte, musste ein Programm entworfen werden, dass diese Funktionalität einbindet und die Konstantendateien einliest.

Da diese Funktionalität später in die Steuerungssoftware der S.RS2 übernommen werden soll, ist es ratsam, die zugrundeliegenden Funktionen so aufzubauen, dass das Einlesen durch den Aufruf einer zentralen Funktion erfolgen kann. Mit `ReadConstFile()` ist diese zentrale Funktion implementiert worden.

Der grobe Ablauf dieser Funktion ist in folgendem Struktogramm dargestellt:




Struktur der Funktion ReadConstFile()

Der vollständige Programmtext der Funktion ist in Anhang A abgedruckt.

Da eXpat ein SAX-Parser ist, wird jedesmal eine vorher zugewiesene Callback-Funktion ausgelöst, wenn der Parser in dem übergebenen Text den Beginn oder das Ende eines Knotens erkennt. Deshalb kommt den Callback-Funktionen eine wichtige Rolle zu: In ihnen wird bestimmt, wie das Programm auf die Knoten reagieren bzw. was es mit den Daten der Knoten anfangen soll.

Die Funktion `XML_TagOpen()` wird ausgelöst, wenn ein öffnendes Tag erkannt wurde. Falls es sich hierbei um einen `Const`-Knoten handelt, wird die entsprechende ID (Konstantennummer) in den Benutzerdaten des Parsers gespeichert. Somit ist eine Zuordnung von nachfolgenden `Item`-Einträgen zu der entsprechenden Konstante möglich.

Falls jedoch `Item`-Einträge erkannt werden, wird zunächst überprüft, ob der Wert mit dem definierten Datentypen übereinstimmt. Ist dies der Fall, so wird eine Funktion aufgerufen, die, in Abhängigkeit von der gespeicherten ID, die Werte an die entsprechenden Stellen in der Steuerung einträgt, indem sie spezielle Funktionen zum Schreiben der Werte aufruft.

Innerhalb dieser Schreib-Funktionen kann nun auf die Konstantenwerte nicht nur über deren Namen, sondern, falls vorhanden, über deren Index zugegriffen werden. Die Funktion `GetItemIndex()` extrahiert den Index aus dem Namen, so dass mit einer `switch()` -Anweisung die Werte der einzelnen Achsen in die Steuerung geschrieben werden können.

Dadurch, dass jede Konstante einzeln untersucht und in einer eigenen Funktion geschrieben wird, ist es möglich, einzelne Konstanten in die Steuerung nachzuladen. Falls die XML-Datei nur drei Konstanten enthält, werden genau diese drei Konstanten auch in die Steuerung geschrieben. Vorhandene Daten werden dabei nicht überschrieben oder auf Default-Werte zurückgesetzt.

Wenn in der Funktion `XML_TagOpen()` andere Knoten als `Const` oder `Item` gefunden werden, werden diese schlicht ignoriert, so z.B. sämtliche Kategorieknoten. Der Steuerung ist es somit egal, ob Konstanten mehrfach in Kategorien referenziert werden oder auch gar nicht referenziert werden. Die Steuerung liest alle `Const`-Knoten, die sie in der Datei findet, genau einmal ein.

Die im Rahmen dieser Arbeit entstandene Funktionalität wurde bisher noch nicht in die eigentliche Steuerungssoftware der S.RS2 integriert. Sie besitzt zunächst vorbereitenden Charakter und kann erst in enger Abstimmung mit den anderen Entwicklern übernommen werden.

Allerdings ist der bisher entstandene Programmcode für VxWorks unter Tornado entwickelt worden und läuft auf der Plattform, auf der auch die eigentliche S.RS2 arbeitet. Eine Übernahme ist somit leicht möglich.

Der vollständige Programmcode, sowohl der, der VxWorks-Implementierung als auch der vom Konstanten-Editor befinden sich auf der beigelegten CD.

Nachfolgend sind alle Funktionen aufgeführt, die im Zusammenspiel die Funktionalität des XML-Konstanten-Lesens implementieren:

`bool CheckConstValue(char *aValue, char *aType)`

Diese Funktion überprüft, ob ein Attributwert mit dem definierten Attributtyp übereinstimmt. Falls ja, wird `true` zurückgegeben.

`char* GetAttrValue(const char* aName, char **attr)`

Diese Funktion ermittelt den Wert des Attributes, das in `*aName` übergeben wird. Ist ein Attribut mit diesem Namen nicht in der Liste `**attr` vorhanden, wird eine leerer Wert zurückgegeben.

`int GetID(const char* aString)`

Diese Funktion ermittelt aus einem ID-String die ID, unter der die Konstante in der Steuerung bekannt ist: `"cst_007zm"` -> `7`.

`int GetItemIndex(char* aName)`

Funktion, die den Index aus einem Item-Namen herausfiltert. Falls kein Index vorliegt, wird der Wert -1 zurückgegeben. (`"Axis[3]"` -> `3`)

`bool isdate(char* aValue)`

Diese Funktion überprüft, ob das übergebene char-Array ein Datum ist. Diese Funktion ist noch nicht implementiert.

`bool isfloat(char* aValue)`

Diese Funktion überprüft, ob das übergebene char-Array eine Fließkommazahl ist.

`bool isfrac(char* aValue)`

Diese Funktion überprüft, ob das übergebene char-Array ein Bruch ist. Diese Funktion ist noch nicht implementiert.

`int isint(char* aValue)`

Diese Funktion überprüft, ob das übergebene char-Array eine Ganzzahl ist.

`int IsItemIndex(char* aName)`

Diese Funktion überprüft, ob ein Item-Name einen Index enthält oder nicht.

`int IsItemName(char* aTest, const char* aName)`

Diese Funktion überprüft, ob ein Item-Name gleich einer Gruppe von Namen entspricht: »Achse[1]« entspricht der Namensgruppe »Achse«. Die in den Klammern [] stehende Zahle stellt hierbei nur den Index der Achse dar.
Parameter: `char* aTest` : Der zu überprüfende Item-Name
`char* aName` : Die gewünschte Namensgruppe

`int ReadConstFile(const char* aFileName, bool Verbose)`

Diese Funktion liest eine XML-Konstantendatei ein und schreibt die darin enthaltenen Werte in die Steuerung. Rückgabewert = 0, falls erfolgreich.

`int SetConst`*`N`*`(char* aName, char* aValue, char* aType)`

Diese Funktionen werden durch `SetConstValue()` aufgerufen. N steht dabei für eine Konstantennummer, z.B. 100. In diesen Funktionen werden die Konstantenwerte in die Steuerung geschrieben.

`int SetConstValue(int ID, char* aName, char* aValue, char* aType)`

Hauptfunktion zum Setzen der Konstanten; hier werden die einzelnen Unterfunktionen zum Zuweisen der Konstanten-Werte in Abhängigkeit der übergebenen ID (= Konstanten-Nummer) aufgerufen.

`char* strdup (char* aString)`

Diese Funktion ist eigentlich in der Standard-Bibliothek <string.h> vorhanden. Leider ist das bei VxWorks nicht der Fall. Die Funktion dupliziert einen String und reserviert dafür Speicher.

`int testit(void)`

Diese Funktion ist der Testaufruf, um eine XML-Konstantendatei einzulesen.

`void XML_TagClose(void* UserData, const char* name)`

Callback-Funktion des XML-Parsers. Sie wird aufgerufen, wenn der XML-Parser das schließende Tag eines Knotens erkannt hat. Parameter:

`*UserData`: Benutzerspezifisch; wird mit XML_SetUserData zugewiesen
`*name`: Name des gefundenen Knotens

`void XML_TagOpen(void *UserData, const char *name, const char **attr)`

Callback-Funktion des XML-Parsers. Sie wird aufgerufen, wenn der XML-Parser das öffnende Tag eines Knotens erkannt hat. Parameter:

`*UserData`: Benutzerspezifisch; wird mit XML_SetUserData zugewiesen
`*name`: Name des gefundenen Knotens
`**attr`: Attributliste des Knotens: `attr[n]` = Attributname,
`attr[n+1]` = Attributwert

# 9 Zusammenfassung und Ausblick

## 9.1 Zusammenfassung

Im Rahmen dieser Arbeit sollte für eine Robotersteuerung eine neue, möglichst universelle Datenschnittstelle entworfen und implementiert werden. Diese Datenschnittstelle sollte in der Lage sein, beliebige, strukturierte Daten im Textformat auszutauschen.

Ein standardisiertes Format, das alle definierten Bedingungen erfüllt, ist XML. Aus diesem Grund wurde eine XML-Schnittstelle in das Realtime-Betriebssystem VxWorks implementiert, das auf der Hardware der Robotersteuerung lauffähig ist.

Um diese Schnittstelle zu verwenden, wurde das bisherige Datenformat zur Übertragung von Roboterkonstanten überarbeitet und in XML neu definiert.

Um mit dem neuen Konstantenformat arbeiten zu können, wurde ebenfalls ein Konstanteneditor entwickelt, der unter Microsoft Windows lauffähig ist und die XML-Konstantendaten lesen, bearbeiten und erzeugen kann.

## 9.2 Ausblick

Zunächst soll die entstandene XML-Schnittstelle in die Robotersteuerung übertragen werden.

Danach bietet es sich an, weitere vorhandene Formate in XML zu definieren. Insbesondere das Folgenformat, das heute bereits an seine Grenzen stößt, kann hier sinnvoll überarbeitet werden.

Wenn weitere Datenformate in XML definiert werden, ist es wünschenswert, hier auch Editoren zu entwickeln, die die Möglichkeit bieten, mit diesen Daten unter Microsoft Windows zu arbeiten.

Insbesondere ein Editor für Folgenformate kann sich sehr umfangreich gestalten. Hier bieten sich neben der textuellen auch grafische Anzeigeoptionen an.

Die bisherige Arbeit bietet eine gute Grundlage, um zukünftig die Robotersteuerung und die zugehörigen Softwaretools modern zu gestalten.

# Anhang A - Wichtige Funktionen der XML-Implementation

```
/* ------------------------------------------------------------------------
 *
 * 2003-08-01 / Zm
 * Callback-Funktion des XML-Parsers. Sie wird aufgerufen, wenn der XML-Parser
 * das oeffnende Tag eines Knotens erkannt hat.
 * Parameter: *UserData : Benutzerspezifisch; wird mit XML_SetUserData zugewiesen
 *            *name     : Name des gefundenen Knotens
 *            **attr    : Attributliste des Knotens: attr[n]   = Attributname,
 *                                                   attr[n+1] = Attributwert
 *
 * ------------------------------------------------------------------------ */
void XML_TagOpen(void *UserData, const char *name, const char **attr)
{
  int *ConstID = (int*)UserData;

  if (strcmp(name, "Const") == 0)
  {
    /* neue Konstante gefunden */
    *ConstID = GetID(GetAttrValue("id", attr));
  }

  else if (strcmp(name, "Item") == 0)
  {
    if (CheckConstValue(GetAttrValue("value", attr), GetAttrValue("type", attr)))
    {
      /* Wert und Typ passen */
      SetConstValue(*ConstID,
                    GetAttrValue("name",  attr),
                    GetAttrValue("value", attr),
                    GetAttrValue("type",  attr));
    }
    else
    {
      /* Wert und Typ passen nicht zusammen */
      printf("Wert (%s) und Typ (%s) stimmen nicht ueberein. Eintrag wird ignoriert!\n",
             GetAttrValue("value", attr),
             GetAttrValue("type", attr));
    }
  }
}



/* ------------------------------------------------------------------------
 *
 * 2003-08-01 / Zm
 * Callback-Funktion des XML-Parsers. Sie wird aufgerufen, wenn der XML-Parser
 * das schliessende Tag eines Knotens erkannt hat.
 * Parameter: *UserData : Benutzerspezifisch; wird mit XML_SetUserData zugewiesen
 *            *name     : Name des gefundenen Knotens
 *
 * ------------------------------------------------------------------------ */
void XML_TagClose(void* UserData, const char* name)
{
  int *ConstID = (int*) UserData;

  if (strcmp(name, "Const") == 0)
    *ConstID = UNKNOWN_ID;
}
```

```
/* ------------------------------------------------------------------------
 *
 * 2003-08-01 / Zm
 * Diese Funktion liest eine XML-Konstantendatei ein und schreibt die darin
 * enthaltenen Werte in die Steuerung.
 * Rueckgabewert = 0, falls erfolgreich
 *
 * ------------------------------------------------------------------------ */
int ReadConstFile(const char* aFileName, bool Verbose)
{
  /* --- vars --- */
  char          strbuf[BUFSIZ];
  unsigned int  readLen;
  int           done;
  int           UserData;
  FILE*         xmlFile;
  XML_Parser    parser;

  /* --- code --- */
  parser = XML_ParserCreate(NULL);

  UserData = UNKNOWN_ID;
  XML_SetUserData(parser, &UserData);
  XML_SetElementHandler(parser, XML_TagOpen, XML_TagClose);

  xmlFile = fopen(aFileName, "r");

  if (xmlFile == 0)
  {
    if (Verbose == true)
      printf("Hat nicht geklappt\n");
    return errno;
  }

  do
  {
    /* XML-Daten aus Datei auslesen */
    readLen = fread(strbuf, 1, sizeof(strbuf), xmlFile);
    done    = readLen < sizeof(strbuf);

    /* Gelesene Daten an den Parser uebergeben */
    if (XML_Parse(parser, strbuf, readLen, done) == XML_STATUS_ERROR)
    {
      if (Verbose == true)
        printf("Fehler beim Parsen!");
      fclose(xmlFile);
      return 1;
    }

  } while (!done);

  fclose(xmlFile);
  XML_ParserFree(parser);

  return 0;
}

/* ------------------------------------------------------------------------
 *
 * 2003-08-01 / Zm
 *
 * ------------------------------------------------------------------------ */
int testit(void)
{
  return ReadConstFile("C:\\TEMP\\test.xml", true);
}
```

## Anhang B - Quellenangaben


[1.1] Richtlinie 98/37 /EG des Europäischen Parlaments und des Rates vom 22. Juni 1998: Kapitel 1, Artikel 1, Absatz 2

[1.2] VDI-Richtlinien 2860: *Montage und Handhabungstechnik*

[1.3] Baginski, Alfredo und Müller, Martin: *InterBus - Grundlagen und Praxis*, 2. Auflage 1998, Hüthig Heidelberg

[1.4] Homepage des W3C: *http://www.w3.org/* , online 2003-08-07

[2.1] Weber, Wolfgang: *Industrieroboter*, 2002, Fachbuchverlag Leipzig im Carl Hanser Verlag, München, Wien

[3.1] Homepage der SEF GmbH: *http://www.sef.de/* , online 2003-08-10

[3.2] Homepage der ADC GmbH: *http://www.adcgmbh.de/* , online 2003-08-10

[3.3] Homepage der Wind River Systems Inc.: *http://www.windriver.com/* , online 2003-08-08

[4.1] SEF Roboter GmbH: *Programmierhandbuch für die SR1-Steuerung*, 1999, Scharnebeck (Programmierhandbuch für die S.RS2 in Vorbereitung)

[5.1] W3C: Veröffentlichung der XML-Spezifikation unter *http://www.w3.org/TR/REC-xml-20001006/* , online 2003-08-07

[5.2] Holzner, Steven: *XML Insider*, 2001, Markt+Technik Verlag, München

[5.3] Kernighan, Brian W. und Ritchie, Dennis M.: *Programmieren in C*, Zweite Auflage 1990, Carl Hanser Verlag, München, Wien

[5.4] Arciniegas, Fabio A.: *XML Developer's Guide*, 2001, Franzis Verlag, Poing

[7.1] Redaktion der Toolbox (Hrsg.): *C++Builder in Team*, 1997, C&L Computer und Literaturverlag, Vaterstetten

[7.2] Kaiser, Richard: *C++ mit dem Borland C++Builder*, 2002, Springer-Verlag, Berlin, Heidelberg

[8.1] W3C: Veröffentlichung der XPath-Spezifikation unter *http://www.w3.org/TR/1999/REC-xpath-19991116/* , online 2003-08-07

[8.2] W3C: Veröffentlichung der DOM-Spezifikation unter *http://www.w3.org/TR/REC-DOM-Level-1/* , online 2003-08-07

[8.3] Homepage des XML-Parsers SAX von David Megginson u.a.: *http://www.saxproject.org/* , online 2003-08-10

[8.4] Homepage vom XML-Parser Apache Xerces (C++ Version): *http://xml.apache.org/xerces-c/index.html* , online 2003-07-30

[8.5] Homepage von James Clarks XML-Parser eXpat: *http://www.jclark.com/xml/expat.html* , online 2003-08-05

[8.6] Wind River Systems Inc.: *Tornado User's Guide*, 1999, Alameda (USA)

## Anhang C - Tabellenverzeichnis

## Anhang D - Abbildungsverzeichnis

Name: Zimmermann
Vorname: Sören
Matr.-Nr.: 151018
Studiengang: Angewandte Automatisierungstechnik

An den Prüfungsausschuss
des Fachbereichs Automatisierungstechnik
der Fachhochschule Nordostniedersachsen
Volgershall 1

21339 Lüneburg

## Erklärung zur Diplomarbeit

Ich versichere, dass ich diese Diplomarbeit - bzw. meinen Teil, der als Gruppenarbeit vergebenen Diplomarbeit - selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Lüneburg, den 2003-08-15

_____