Fachbereich 3

Diplomarbeit
im Fach Informatik

SPS-Emulator
für den Einsatz in der Lehre

vorgelegt von

Lars Struß <lst@tzi.de>
Matrikelnummer: 1450662

September 2005

Gutachter:

Prof. Dr. F. W. Bruns

Prof. Dr. K.-H. Rödiger

Inhaltsverzeichnis

1. Aufgabenstellung

Zur Produktion von Gütern aller Art werden in den modernen Industrieländern schon seit Jahren hoch technisierte Maschinen eingesetzt, die immer umfangreichere und komplexere Arbeitsabläufe übernehmen. Mit zunehmender Komplexität der Arbeitsabläufe steigen auch die Anforderungen an die Steuerung einer Maschine. Sie muss alle Baugruppen koordinieren, Fehler erkennen und Techniker bei Wartung und Fehlerdiagnose unterstützen.

Eine der wichtigsten Steuerungstechnologien zur Prozessautomatisierung ist die *speicherprogrammierbare Steuerung* (SPS; siehe [Wiki01]). Im einfachsten Fall enthält sie einen Mikroprozessor als zentrale Steuerinstanz, einen Speicherbaustein für das Programm und Ein-/Ausgabe-Schnittstellen zur Interaktion mit der Außenwelt. Moderne SPS entwickeln sich zunehmend zu vollwertigen Computern mit Netzwerkfunktionen, Displays und umfangreichen Fernwartungsmöglichkeiten. Somit treffen hier unterschiedlichste Forschungsdisziplinen aufeinander. Viele Begriffe aus der SPS-Technik haben ihren Ursprung in der klassischen Elektrotechnik und in der Informatik.

Die zunehmende Komplexität der Automatisierungssysteme erfordert auch neue Wege bei der Entwicklung ihrer Hard- und Software. Hier gewinnen zunehmend die sogenannten Virtualisierungskonzepte an Bedeutung. Mit diesem Verfahren lassen sich insbesondere Entwicklungszeiten verkürzen und Entwicklungsrisiken minimieren. [Müller04] nennt hier z.B. das Virtual-Manufacturing, bei dem im Computer simulierte Baugruppen zu einem virtuellen Produkt zusammengefügt werden und auf diese Weise bereits vor der realen Herstellung des Produktes erste Tests durchgeführt werden können.

Zwischen der realen und der virtuellen Entwicklung eines Produktes steht die sogenannte Mixed Reality, die reale und virtuelle Bausteine zu einem hybriden Ganzen zusammenfügt. Eine ausführliche Begriffsdefinition findet sich in [Müller02].

Diesen neuartigen Anforderungen gilt es auch in der Lehre Rechnung zu tragen. Die Ausbildung sollte eine virtuelle Lernumgebung bieten, in der Lernende gefahrlos experimentieren kann, muss jedoch auch eine Ausbildung mit realen Maschinen beinhalten, um dem Auszubildenden ein Gefühl für den Umgang mit Maschine und Werkstück zu geben (vgl. [Müller02]).

Ich möchte Studenten/Auszubildenden den Einstieg in diese komplexe Materie erleichtern, indem ich mit dieser Diplomarbeit eine SPS-Lernplattform schaffe, die einen kostengünstigen und ungefährlichen ersten Zugang zur Automatisierungstechnik bietet.

Die vorliegende Diplomarbeit besteht aus dem Programmpaket „SPS-Lernplattform“ sowie dem vorliegenden Dokument.

Die Gliederung dieses Dokuments orientiert sich an einem Softwareprojekt. Es beginnt mit einer Anforderungsdefinition und endet mit der Dokumentation der Implementierung.

2. Anforderungsdefinition für eine SPS-Lernplattform

Ziel dieser Diplomarbeit ist die Schaffung einer Lernplattform, die es den Studenten ermöglicht, sich zu Hause und am Ausbildungsplatz praktisch mit der SPS-Programmierung auseinander zu setzen. Ausgehend von diesem Gedanken ergibt sich die erste Anforderungsdefinition.

2.1. Vorüberlegungen

Nachdem im vorangegangenen Kapitel die Anforderungen und Wandlungen in der Ausbildung skizziert wurden, muss nun die Frage gestellt werden, wie ein solcher Ausbildungsbetrieb im Bereich Automatisierungstechnik sinnvoll ablaufen kann.

Der Ansatz, der einem industriellen Einsatz am nächsten käme, jedem Auszubildendem eine reale SPS zur Verfügung zu stellen, scheitert nicht nur an den hohen Beschaffungskosten der SPS-Entwicklungspakete. Eine SPS kann nur sinnvoll eingesetzt werden, wenn auch ein flexibel zu steuerndes System existiert, denn mit simplen Ein-/Ausgabemöglichkeiten können nur schwerlich komplexe Steuerungen entwickelt werden. Reale Anlagen wie beispielsweise die Lernsysteme der Firma Festo (u.a. MPS) sind jedoch auch wiederum mit hohen Kosten verbunden. Eine gängige Lösung dieses Dilemmas ist Lernsoftware in Gestalt von Simulatoren, die alle teuren und gefährlichen Komponenten virtualisieren und als digitales Computermodell bereitstellen.

Wissenschaftliche Untersuchungen derzeit verfügbarer Lernsoftware wie z.B. in [Severing01] zeigen jedoch noch erhebliche Defizite auf.

So kritisiert [Severing01], dass
- viele Lernprogramme lediglich mediale Aufbereitungen vorhandener Lehrbücher seien, die die Vorteile der neuen Technik nicht nutzen.
- Lernprogramme meistens proprietäre Systeme mit proprietären Dateiformaten seien, deren Pflege selbst für den Hersteller mit hohen Kosten verbunden sei.
- Lernprogramme oft Insellösungen sind, die sich nicht nahtlos in die betrieblichen Abläufe eines Betriebs einbinden ließen.

Dies lässt sich zur Anforderung der Offenheit an ein Lernsystem zusammenfassen. Eine ideale Lernplattform lässt sich an vorhandene Hard- und Software anpassen und unterstützt so den Mixed-Reality-Gedanken: Der Auszubildende kann zunächst eine Maschine vollständig virtuell im Computer entwickeln, programmieren und testen. Anschließend können schrittweise immer mehr virtuelle Komponenten durch reale ersetzt werden bis am Ende eine reale Maschine entsteht.

2.2. Anforderungsdefinition für eine SPS-Lernumgebung

2.2.1. Grundlegende Anforderungen

Das System sollte

- sich am AWL-Befehlssatz der Siemens S7-Baureihe orientieren, da diese Programmiersprache einen De-facto-Standard für SPS-Programme darstellt und auch von vielen anderen Herstellern unterstützt wird.

- keine besondere Hardware erfordern, sondern auf handelsüblichen Computern funktionieren. Diese werden von den meisten Studenten ohnehin benötigt und sind daher in der Regel privat oder in Rechnerpools an der Ausbildungsstelle vorhanden.
- ohne Lizenzkosten jedem Studenten zur Verfügung gestellt werden können (freeware).
- plattformunabhängig sein, um die Systemvielfalt in der freien Lehre zu unterstützen.
- als Open-Source-Projekt von Studenten und Lehrenden an besondere Einsatzzwecke angepasst werden können.
- die reale Hardware möglichst originalgetreu simulieren.
- möglichst modular strukturiert sein um eine bestmögliche Anpassung an viele verschiedene Aufgabenfelder zu ermöglichen.
- möglichst echtzeitfähig sein.

2.2.2. Praxisbezogene Anforderungen

2.2.2.1. Präzisierung der Anforderungen

Um diese grundlegende Anforderungsdefinition noch besser an die Erfordernisse der Ausbildungspraxis anzupassen, bat ich Fachleute um eine Einschätzung der wesentlichen Anforderungen an eine Lernplattform.

Bei den Fachleuten handelte es sich um einen erfahrenen Automatisierungstechnik-Lehrer, einen Mechatronik-Wissenschaftler und Ingenieur, einen Produktionsinformatiker sowie einen Produktionsinformatiker und Mixed Reality Experten. Diese wurden ausgewählt, da sie als potentielle Nutzer der Lernumgebung in Betracht kommen und ihre Bedürfnisse deswegen besonders berücksichtigt werden sollten.

Die Umfrage bestand aus zwei Teilen. Im ersten Teil sollten eine Reihe von allgemeinen SPS-Funktionen nach Wichtigkeit geordnet werden. Im zweiten Teil ging es darum, Befehlsgruppen nach der subjektiven Wichtigkeit zu sortieren. Die Befehlsgruppen orientieren sich dabei an den Kategorien der Siemens Befehlsreferenz „Anweisungsliste (AWL) für S7-300&400“.

Teil 1: Allgemeine Funktionalität
- umfangreicher AWL-Befehlssatz (siehe auch nächste Frage) - symbolische Programmierung mit benannten Variablen anstelle von direkten Hardwareadressen/Merkern - einfache Funktionsbausteine (keine oder einfache Parameterübergabe) - komplexe Funktionsbausteine (Multi-Instanzen) - Bibliothek mit Standard-Funktionsbausteinen - vielfältige Beispielapplikationen zum Verbinden mit dem Server - Lösungsprogramme zu den Beispielapplikationen - Steuerung realer Hardwarekomponenten (z.B. Treiber für Sensorik/Aktorik am LPT-Port) - besonders umfangreiche Programmdokumentation, Lernprogramme, Assistenten - hohe Übereinstimmung der Emulation mit der realen Hardware in Detailfragen

Teil 2: AWL-Befehlssatz
Unterstützung von - Sprungbefehlen wie SPA, SPL, SPB, ...

- grundlegenden Akkuoperationen (Vergleichen, einfache Rechenoperationen, Laden/Transferieren)
- erweiterten Akkuoperationen (mathematische Funktionen wie sin, cos, tan, ...)
- Umwandlungsfunktionen (BTI, ITB, BTD, DTB, ...)
- Zählern
- Zeitgebern (Timer)
- Datenbaustein-Operationen (AUF, TDB, L DBLG, ...)
- Gleitpunktzahlen (bei Akkuberechnungen)
- Zeigeroperationen (Adressregister)
- Master-Control-Relay (MCR(,)MCR, MCRA, MCRD)

Da die Unterstützung der grundlegenden Logikbefehle (z.B. U, O, X) absolut unverzichtbar ist, wurden sie als selbstverständlich vorausgesetzt und bei den Fragen nicht berücksichtigt.

Von den vier angeschriebenen Experten antworteten drei mit einer Priorisierung der Funktionsliste. Leider wurde bei der Beantwortung die Funktionsliste nicht in eine Reihenfolge mit absteigender Priorität gebracht, sondern den einzelnen Funktionen Prioritätszahlen ohne feste Skala zugeordnet wurden. Dies erschwert die Auswertung, die nun nur noch relativ erfolgen kann.

Um dennoch zu einem verwendbaren Ergebnis zu kommen, wurden alle Prioritäten auf das Intervall von 0 bis 1 normiert und anschließend gemittelt. Das Ergebnis ist in der Tabelle auf der nächsten Seite aufgelistet, wobei der Wert 0 der höchsten Priorität entspricht. Anforderungen mit der Priorität 0 haben den Rang 1, höhere Prioritätszahlen führen zu einer höheren Rangzahl, wobei gleiche Prioritätszahlen den gleichen Rang bekommen.

Beispiel:
Bei der Antwort von Experte 1 wurden beim ersten Fragenblock (Teil 1) die Prioritäten 1 bis 3 vergeben, die größte Prioritätszahl ist also 3. Der Bereich 1 bis 3 (genannte Prioritäten) muss auf 0 bis 1 (normierte Prioritäten) abgebildet werden.

Tabelle 1: Ergebnis der Umfrage

Bereich:	Rang:	Anforderung:	Mittelwert:	Antwort Experte 1:	Normiert:	Antwort Experte 2:	Normiert:	Antwort Experte 3:	Normiert:
Teil 1	1	einfache Funktionsbausteine (keine oder einfache Parameterübergabe)	**0,0**	1	0,00	1	0,00	1	0,00
	2	Steuerung realer Hardwarekomponenten (z.B. Treiber für Sensorik/Aktorik am LPT-Port)	**0,2**	1	0,00	2	0,25	1	0,33
	2	umfangreicher AWL-Befehlssatz (siehe auch nächste Frage)	**0,2**	1	0,00	1	0,00	2	0,67
	3	hohe Übereinstimmung der Emulation mit der realen Hardware in Detailfragen	**0,3**	1	0,00	2	0,25	2	0,67
	4	Bibliothek mit Standard-Funktionsbausteinen	**0,5**	2	0,50	4	0,75	1	0,33
	5	symbolische Programmierung mit benannten Variablen anstelle von direkten Hardwareadressen/Merkern	**0,6**	3	1,00	1	0,00	2	0,67
	5	besonders umfangreiche Programmdokumentation, Lernprogramme, Assistenten	**0,6**	1	0,00	4	0,75	3	1,00
	6	komplexe Funktionsbausteine (Multi-Instanzen)	**0,7**		1,00	3	0,50	2	0,67
	7	vielfältige Beispielapplikationen zum Verbinden mit dem Server	**0,9**		1,00	5	1,00	2	0,67
	7	Lösungsprogramme zu den Beispielapplikationen	**0,9**		1,00	5	1,00	2	0,67
		Max:		*3*		*5*		*3*	
Teil 2	1	Unterstützung von Zählern	**0,0**	1	0,00	1	0,00	1	0,00
	1	Unterstützung von Zeitgebern (Timer)	**0,0**	1	0,00	1	0,00	1	0,00
	2	Unterstützung von Sprungbefehlen wie SPA, SPL, SPB, ...	**0,3**	2	0,33	1	0,00	2	0,50
	3	Unterstützung von grundlegenden Akkuoperationen (Vergleichen, einfache Rechenoperationen, Laden/Transferieren)	**0,5**	4	1,00	1	0,00	2	0,50
	4	Unterstützung von Gleitpunktzahlen (bei Akkuberechnungen)	**0,6**	4	1,00	2	0,33	2	0,50
	5	Unterstützung von Datenbaustein-Operationen (AUF, TDB, L DBLG, ...)	**0,7**	4	1,00	3	0,67	2	0,50
	5	Unterstützung von Zeigeroperationen (Adressregister)	**0,7**	4	1,00	3	0,67	2	0,50
	6	Unterstützung von erweiterten Akkuoperationen (mathematische Funktionen wie sin, cos, tan, ...)	**0,8**	4	1,00	2	0,33	3	1,00
	6	Unterstützung von Umwandlungsfunktionen (BTI, ITB, BTD, DTB, ...)	**0,8**	4	1,00	2	0,33	3	1,00
	6	Unterstützung des Master-Control-Relay (MCR(,)MCR, MCRA, MCRD)	**0,8**		1,00	4	1,00	2	0,50
		Max:		*4*		*4*		*3*	

2.2.2.2. Bewertung der Anforderungen

Funktionsbausteine sind ein wichtiges Sprachelement zur Strukturierung des Programmcodes. Sie ermöglichen außerdem die Wiederverwendung des Codes. Ein umfangreicher Befehlssatz, Zähler und Zeitgeber bilden zudem das nötige Reservoir um Problemstellungen effizient, d.h. kurz und performant, lösen zu können. Wichtig ist auch eine hohe Übereinstimmung zwischen realer SPS und emulierter SPS, um die im Emulator gewonnenen Erkenntnisse später auf reale Anlagen übertragen zu können. Die Emulation sollte jedoch auf einer abstrakten Ebene bleiben, die keine spezifische SPS (möglicherweise sogar mit ihren Fehlern in der Firmware) kopiert, sondern einer Gruppe von SPS gerecht wird und eine idealisierte SPS repräsentiert.

Ein Interfacetreiber zur Steuerung realer Hardwarekomponenten ist eine nützliche Funktion, allerdings ist sie kaum ohne die Beschränkung der Allgemeinheit des Programms umsetzbar. Es gibt eine Vielzahl von Interfaces, die über eine Vielzahl von Schnittstellen (seriell/parallele Schnittstelle, USB, PCI-Steckkarte, Ethernet) an den Computer angeschlossen werden. Eine solche Hardwareunterstützung erfordert also einen eigenen Treiber für jedes einzelne Hardwareinterface auf jeder einzelnen Plattform, was die Universalität der Software stark einschränkt. Ich beschränke die Anforderung daher auf die Entwicklung eines Frameworks, das eine plattformunabhängige Schnittstelle zur SPS-Lernumgebung bereitstellt und einfach um die interface- und plattformspezifischen Programmteile erweitert werden kann.

Die Bibliothek mit komplexen Standardfunktionen betrachte ich als optionales Feature, da sie sich auch noch nachträglich erstellen lässt. Rechenfunktionen können mit Hilfe des vorhandenen Befehlssatzes nachgebildet werden. Bei den Systemfunktionen handelt es sich häufig um spezielle Funktionen zur Programmierung spezieller Hardwarebaugruppen wie einer Echtzeituhr oder Erweiterungen zur Systemüberwachung, deren Implementierung den Rahmen dieser Diplomarbeit sprengen würde.

Symbolische Programmierung ist sinnvoll, da sie die Programmierung allgemeiner, also bibliotheksfähiger Bausteine ermöglicht, und hilft, Ressourcenkonflikte um einzelne Merkeradressen zu vermeiden. Symbolische Programmierung erfordert jedoch nicht notwendigerweise einen grafischen Symboltabelleneditor, es ist ausreichend, die benutzten Symbole wie Variablen und Konstanten textuell im Quelltext zu deklarieren.

Da vielfältige Beispielapplikationen mit Lösungsprogrammen nicht gefragt sind, sind ein oder zwei Beispielprogramme mit exemplarischem Steuerprogramm sicherlich ausreichend. Vollständige Lösungsprogramme wären ohnehin nur sinnvoll, wenn sie nicht frei erhältlich wären und somit nicht zum Schummeln bei der Aufgabenstellung benutzt werden können. Diese eingeschränkte Verbreitung steht jedoch im Widerspruch zum Open Source-Gedanken, weshalb ich darauf verzichte.

Die Unterstützung höherer mathematischer Funktionen kann als Option angesehen werden, da SPS nur selten zur Durchführung komplizierter Berechnungen benutzt werden. Umwandlungsfunktionen, z.B. zur Konvertierung von Dualzahlen in BCD-Notation, sind jedoch nicht nur zur Ansteuerung externer Displays sehr hilfreich.

Zusammenfassend ergibt sich die folgende Bewertung der einzelnen Anforderungen:

Anforderung:	Bewertung:
einfache Funktionsbausteine (keine oder einfache Parameterübergabe)	wichtig
Steuerung realer Hardwarekomponenten (z.B. Treiber für Sensorik/Aktorik am LPT-Port)	Framework wichtig Treiber optional
umfangreicher AWL-Befehlssatz (siehe auch nächste Frage)	wichtig
hohe Übereinstimmung der Emulation mit der realen Hardware in Detailfragen	wichtig
Bibliothek mit Standard-Funktionsbausteinen	optional
symbolische Programmierung mit benannten Variablen anstelle von direkten Hardwareadressen/Merkern	wichtig, aber reduzierte Variante
besonders umfangreiche Programmdokumentation, Lernprogramme, Assistenten	optional
komplexe Funktionsbausteine (Multi-Instanzen)	optional
vielfältige Beispielapplikationen zum Verbinden mit dem Server	optional (nur 1-2 Beispielapplikationen)
Lösungsprogramme zu den Beispielapplikationen	entfällt
Unterstützung von Zählern	wichtig
Unterstützung von Zeitgebern (Timer)	wichtig
Unterstützung von Sprungbefehlen wie SPA, SPL, SPB, ...	wichtig
Unterstützung von grundlegenden Akkuoperationen (Vergleichen, einfache Rechenoperationen, Laden/Transferieren)	wichtig
Unterstützung von Gleitpunktzahlen (bei Akkuberechnungen)	optional
Unterstützung von Datenbaustein-Operationen (AUF, TDB, L DBLG, ...)	optional
Unterstützung von Zeigeroperationen (Adressregister)	optional
Unterstützung von erweiterten Akkuoperationen (mathematische Funktionen wie sin, cos, tan, ...)	optional
Unterstützung von Umwandlungsfunktionen (BTI, ITB, BTD, DTB, ...)	wichtig
Unterstützung des Master-Control-Relay (MCR(,)MCR, MCRA, MCRD)	optional

2.2.3. Anwendungsszenarien

Mit den zuvor genannten Anforderungen lassen sich zum Beispiel die folgenden Anwendungsszenarien umsetzen:

2.2.3.1. Einfache SPS-Simulation

Der Student installiert den Emulator-Kern und die Steuerkonsole auf seinem PC. Er schreibt ein SPS-Programm, startet es und setzt manuell über die Steuerkonsole einzelne Eingänge an der virtuellen SPS. Dabei beobachtet er das Verhalten der Ausgänge, um die Interpretation des Programms in der SPS nachzuvollziehen.

2.2.3.2. Programmierung einer SPS mit lokaler virtueller Applikation

Der Student installiert den Emulator-Kern, die Steuerkonsole und die Fahrstuhl-Applikation auf seinem Computer. Er verbindet das simulierte Fahrstuhlmodell mit der simulierten SPS und schreibt ein SPS-Programm, das den Fahrstuhl entsprechend der Tasteneingabe in den einzelnen Etagen steuert.

2.2.3.3. Programmierung eines Mixed-Reality-Systems mit verteilten Funktionen

Bei diesem Szenario wird eine Hybridlösung eingesetzt. Sie besteht aus einem Modell eines Fahrstuhls aus Fischertechnik. Die Sensoren und Aktoren des Modells werden an eine PC-Schnittstellenkarte angeschlossen und diese mit dem Hardware-Interface-Treiber an eine virtuelle SPS auf einem Uni-Computer angebunden. Eine Webcam beobachtet das Fischertechnikmodell und sendet die Bilder ins Internet.
Ein Student installiert sich den Emulator-Kern, die Steuerkonsole und die Fahrstuhl-Applikation auf seinem PC. Er programmiert ein SPS-Programm, das die Etagentaster des virtuellen Fahrstuhlmodells abfragt und das reale Fischertechnikmodell ansteuert. Zum Testen verbindet er sich über das Internet mit der virtuellen SPS und betrachtet das Modell über die Webcam. Da sich die virtuelle SPS und das Hardwaremodell in der Uni befinden, müssen nur die weniger zeitkritischen Steuerbefehle über die Internetleitung übertragen werden. Die reale Hardware ist über das schnelle Uni-Netzwerk an die virtuelle SPS angebunden.

2.2.3.4. Programmierung einer virtuellen Applikation mit einer realen SPS

Der Student lädt ein Programm in die virtuelle SPS, das alle Eingänge auf die jeweiligen Ausgänge abbildet (A0.0←E0.0, A0.1←E0.1, A0.2←E0.2, ...). Dann verbindet er die Fahrstuhlapplikation und eine Hardware-Interface-Applikation mit der virtuellen SPS. An das Hardware-Interface schließt er eine reale SPS an und steuert darüber die Fahrstuhlapplikation. (vgl. hierzu auch Festo EasyVeep [Festo01])

2.2.3.5. Fernüberwachung von Signalpegeln

Der Student baut in der Uni einen Langzeitversuch auf und verbindet dafür Sensoren über ein Hardware-Interface mit der virtuellen SPS. Während der Versuchsdauer überwacht er auf seinem Computer zu Hause über die Steuerkonsole oder eine selbstgeschriebene Anzeigeapplikation mit Plotterfunktion/Langzeitspeicherung die Messdaten.

Um Wiederholungen zu vermeiden, verzichte ich auf eine zusätzliche Detailspezifikation. Details können der oben beschriebenen Anforderungsspezifikation entnommen werden.

3. Designspezifikation der SPS-Lernplattform

Die folgenden Kapitel 3.1. und 3.2. beschreiben zunächst die verschiedenen Alternativen, die ich beim Entwurf der Lernplattform berücksichtigt habe. In den darauf folgenden Kapitel sind die hieraus abgeleiteten Entscheidungen dargelegt.

3.1. Überblick über bereits existierende Tools und Vergleich mit der Anforderungsdefinition

Ich habe im Rahmen dieser Diplomarbeit verschiedene Programme unter den beiden folgenden Fragestellungen analysiert:

1. Welche Simulationslösungen bereits existieren bereits?
2. Gibt es Programme, die die im vorherigen Kapitel beschriebenen Anforderungen abdecken und sich als Basis für eine SPS-Lernplattform anbieten?

Die meiste der von mir untersuchten Software ist kommerzieller Natur. Sie wurde für den professionellen Einsatz entwickelt. Der Einsatz in der Lehre ist aus Kosten- und Lizenzierungsgründen schwierig.

Auf drei Programme, deren Einsatz in der Lehre möglich ist, möchte ich kurz eingehen:

3.1.1. SPS4Linux

SPS4Linux [Eilers01] ist die Linux-Portierung eines 10 Jahre alten DOS-Programms. Es unterstützt Anweisungslisten mit einfachen Befehlen und gestattet die Steuerung von externer Hardware über die parallele Schnittstelle. Auf der Homepage des Programms gibt es Schaltpläne für Beispielschaltungen. Die Bedienoberfläche ist einfach und erinnert stark an Turbo Pascal. Programm und Pascal-Quellcode sind verfügbar.

3.1.2. WinSPS-S7

WinSPS-S7 [MHJ01] ist ein leistungsfähiges kommerzielles Softwarepaket zum Simulieren und Programmieren von Siemens S7-SPS. Das Programm unterstützt auch komplizierte Anweisungslisten (AWL-Programme) sowie Funktions- und Kontaktplandarstellungen. Das Programm verfügt im großen und ganzen über eine professionelle grafische Benutzeroberfläche. Die Eingabe von AWL-Code ist mit dem eingebauten Editor jedoch umständlich und nicht sehr intuitiv. Die kommerzielle Version ist teuer. Es gibt jedoch eine eingeschränkte Demoversion, deren Zeitbeschränkung durch einen über die Homepage kostenlos anforderbaren Registriercode aufgehoben werden kann. Alle anderen Beschränkungen bleiben dabei bestehen.

3.1.3. MatPLC

MatPLC [MatPLC01] ist ein Open-Source-Projekt (GPL-Lizenz) zur Entwicklung von Steuerungsapplikationen auf unterschiedlicher Hardware. Das Programm ist modular aufgebaut, so dass man nicht auf eine bestimmte Programmiersprache oder Ein-/Ausgabe festgelegt ist. Es gibt auch einen IEC ST/IL Compiler, der nach Aussage des Handbuchs jedoch noch sehr unvollständig ist. Insgesamt macht das Projekt noch einen recht unvollständigen Eindruck.

3.1.4. Fazit

WinSPS-S7 ist das mit Abstand leistungsfähigste und ausgereifteste Programm im Vergleich. Es stellt beinahe selbst eine eigenständige SPS-Lernplattform dar. Die hohen Lizenzgebühren und die Nichtverfügbarkeit des Quellcodes verhindern jedoch einen freien Einsatz des Programms in der Lehre und schränken die Erweiterungsmöglichkeiten (z.B. Applikationen zur Simulation weiterer Hardware) drastisch ein. Die Software ist zudem nur für Windows verfügbar.

SPS4Linux läuft, wie der Name schon sagt, nur unter Linux. Pascal als verwendete Programmiersprache macht die Portierung auf exotischere Hardware schwierig und der implementierte minimalistische Befehlssatz lässt sich nur schwierig erweitern.

MatPLC wäre die geeigneteste Basis für eine SPS-Lernplattform. Die Entwickler der Software legen großen Wert auf Modularität, so dass eine Erweiterung des Programms denkbar wäre. Gegen MatPLC spricht jedoch das noch sehr frühe Entwicklungsstadium der Software. Laut Programmhomepage befindet sich die Software noch im Alphastadium. Viele Module sowie die Dokumentation sind noch sehr unvollständig. Die Entwicklung scheint nur schleppend voranzugehen. Es stellt sich zudem die Frage, ob MatPLC nicht zu leistungsfähig ist und einen Einsteiger in die SPS-Technik überfordert.

Ich habe mich daher entschlossen, keines dieser Programme als Basis zu verwenden und die Software selbst zu entwickeln.

3.2. Erörterung von Designalternativen

3.2.1. Programmiersprache

Zur Implementierung bieten sich die folgenden Programmiersprachen an:

- C/C++
- C# (Microsoft .NET)
- Java

C/C++ ist die wohl gängigste Programmiersprache, die von jeder ernst zu nehmenden Plattform unterstützt wird. Sie bietet eine hohe Performance bis hin zur Echtzeitfähigkeit, da der Quellcode direkt in Maschinencode für die jeweilige CPU übersetzt wird. Nachteilig ist, dass echte plattformübergreifende Programmierung nur mit zusätzlichen Bibliotheken möglich ist, die sich als Zwischenschicht zwischen Anwendung und Betriebsystem schieben.

C# und Java haben diese Abstraktionsebene bereits in ihrer Laufzeitumgebung eingebaut. Das Programm läuft dadurch immer in einer bekannten Umgebung mit definierten Eigenschaften ab. In C# ist diese Entkopplung von der zugrundeliegenden Plattform nur halbherzig gelöst, da Microsoft die .NET-Umgebung nur für die hauseigene Windows Produktfamilie anbietet. Eine offene Laufzeitumgebung für andere Systeme ist derzeit als Open-Source-Projekt unter [Mono01] in der Entwicklung.

Java ist von Haus aus für eine Vielzahl von Plattformen verfügbar. Zudem ist eine reduzierte Version für Geräte mit geringer CPU-Leistung und kleinem Speicher verfügbar. Dies vereinfacht eine spätere Anpassung der Lernumgebung an PDAs oder Smartphones, falls dies gewünscht ist. Auf diese Weise kann z.B. ein kleiner mobiler PDA einen kleinen Roboter steuern. Des Weiteren existiert für die Programmiersprache Java eine breite Anzahl freier Compiler, Entwicklungsumgebungen und Zusatzbibliotheken.

Sowohl C# (bzw. Microsoft .NET) als auch Java ist gemeinsam, dass die Laufzeitumgebung das Programm z.T. stark verlangsamt und die Programme nur eingeschränkt echtzeitfähig sind.

3.2.2. Interpreter vs. Compiler

Zur Ausführung des AWL-Programmcodes sind zwei grundsätzliche Verfahren denkbar, der Interpreter und der Compiler.

Ein *Interpreter* liest eine Zeile des Quellcodes und entschlüsselt aus deren Inhalt, was das zu interpretierende Programm in dieser Zeile tun soll. Der Interpreter führt die erforderlichen Operationen anschließend stellvertretend aus. Ein Interpreter ermöglicht die Ausführung des interpretierten Programms auf jeder Hardware, auf der der Interpreter lauffähig ist. Diesem Vorteil steht eine schlechte Performance gegenüber, denn jede Programmzeile muss jedes Mal, wenn sie ausgeführt werden soll, aufs Neue in ihre syntaktischen Bestandteile zerlegt und analysiert werden.

Auch *Compiler* analysieren die Programmzeilen. Die Analyse erfolgt jedoch in einem Schritt. Aus dem Analyseergebnis wird ein Maschinencode-Programm erzeugt, welches sich direkt auf einem bestimmten Prozessor unter einem bestimmten Betriebsystem ausführen lässt. Soll das Programm unter einer anderen Rechnerarchitektur ausgeführt werden, muss zunächst ein für diese Rechnerarchitektur passender Compiler vorhanden sein.

3.2.3. Integrierte Applikation vs. Client/Server-Modell

Bei der Programmstruktur stellt sich die Wahl zwischen einem großen Programm, das alle Programmteile in einer ausführbaren Datei zusammenfasst und einem dezentralen Client-Server-Modell, bei dem alle Programmmodule eigenständige Programme sind, die über ein Netzwerk miteinander kommunizieren.

Für eine integrierte Applikation sprechen die folgenden Punkte:
- Die Bedienung ist leichter, da kein Netzwerk konfiguriert werden muss. Alles besteht „aus einem Guss“.
- Die Latenzzeiten sind gering, da alle Befehle sofort ausgeführt werden können und der Umweg über das vergleichsweise langsame Netzwerk entfällt.
- Die Deterministik ist besser.

Für ein dezentrales Client/Server-Modell spricht, dass
- Funktionsgruppen klar getrennt sind und es keine Vermischung von Applikation, SPS und Programmierumgebung gibt.

- eine sehr gute Erweiterbarkeit durch Zusatzmodule möglich ist, auch über Programmiersprachengrenzen hinweg.
- bei Bedarf die Verteilung der Programmteile auf verschiedene Rechner möglich ist, z.B. zur Verteilung der Rechenlast oder bei geografischer Trennung von Anlagenteilen oder Nutzern.

3.2.4. RPC/RMI vs. eigenes Netzwerkprotokoll

Entscheidet man sich im vorherigen Abschnitt für ein Client/Server-Modell, so muss die Art der Kommunikation zwischen den Programmmodulen spezifiziert werden. Auch hier bieten sich zwei grundlegende Möglichkeiten an:

- die Nutzung eines vorhandenen RPC (Remote Procedure Call)-Protokolls oder
- die Entwicklung eines eigenen Protokolls auf Basis von
    - Punkt-zu-Punkt-Kommunikation (Unicast) oder
    - Mehrpunkt-Kommunikation (Multicast)

RPC-Dienste dienen zum Aufruf von Funktionen/Methoden auf entfernten Rechnern. Es ist somit möglich, einen Programmteil auf einem fremden Computer -im Rahmen gewisser Einschränkungen wie z.B. Zugriffsbeschränkungen- so aufzurufen, als wäre das entfernte Programm ein ganz normales Unterprogramm. Die Programmiersprache Java hat einen RPC-Dienst in Form der RMI (Java Remote Method Invocation, [Sun03]) bereits fertig integriert. Als Programmiersprachenübergreifende RPC-Dienste/-protokolle seien an dieser Stelle noch CORBA und SOAP genannt.

Allen RPC-System ist gemeinsam, dass sie für ein sehr breites Anwendungsspektrum entworfen wurden und sich ein mächtiger Funktionsumfang in der Komplexität und dem Overhead des Protokolls niederschlägt. Ein selbstentwickeltes Protokoll bietet hier den Vorteil, dass dieses Protokoll optimal an den Anwendungsfall angepasst werden kann, so dass die Datenübertragung simpel und latenzfrei bleibt.

Prinzipiell kann ein selbstentwickeltes Protokoll Punkt-zu-Punkt- oder Mehrpunkt-Verbindungen aufbauen. Mehrpunkt-Verbindungen (sogenannte Multicasts) reduzieren die Netzwerkbelastung wenn identische Daten an viele Empfänger verteilt werden sollen. Der Sender sendet dabei nicht wie bei einer Punkt-zu-Punkt-Übertragung an genau einen Empfänger, sondern an eine Gruppe von Empfängern. Zur Verwaltung der Gruppenzugehörigkeiten sind jedoch u.a. spezielle multicastfähige Netzwerkkomponenten erforderlich, die noch nicht im gesamten Internet vorhanden sind. Ein weiteres Problem bei Multicast-Übertragungen ist die Fehlerkorrektur bei Paketverlusten, die nur mit hohem Aufwand sichergestellt werden kann.

3.2.5. Festlegungen

Um die gewünschten Anforderungen in die Praxis umzusetzen, habe ich mich für ein modulares Programm in der Programmiersprache Java entschieden. Java-Applikationen laufen auf den wichtigsten der heute üblichen Computer. Dadurch ist der Nutzer nicht gezwungen, sich mehrere Betriebsysteme oder gar mehrere Rechnerarchitekturen (x86, PowerPC) zu installieren. Er kann im Rechnerraum der Schule/Universität den nächsten

verfügbaren Computer verwenden, muss somit nicht auf einen evtl. knappen Platz an einem Rechner mit bestimmter Architektur warten. Außerdem steigt die durchschnittliche Leistungsfähigkeit der Computer rapide an, so dass die Geschwindigkeitseinbußen einer interpretierten Programmiersprache zunehmend zu vernachlässigen sind. Da die meisten Anwendungen im Bereich der Lehre eher auf Grundlagen aufbauen und nicht zeitkritisch sind, ist dieser Kompromiss bei der Echtzeitfähigkeit der Software akzeptabel.

Zur Ausführung der AWL-Programme habe ich mich für eine Hybridlösung aus Compiler und Interpreter entschieden. Ein reiner Interpreter erfordert zuviel Rechenzeit, da die Analyse der Quellcodezeilen sehr aufwendig ist. In einer ohnehin schon vergleichsweise langsamen Programmiersprache sollte die vorhandene Rechenzeit also so effizient wie möglich eingesetzt werden. Ein reiner Compiler kommt für SPS-Lernplattform jedoch auch nicht in Frage, da er, wie oben ausgeführt, sehr stark mit der zugrunde liegenden Hardware verzahnt und somit kaum mit dem Entwicklungsziel der Plattformunabhängigkeit vereinbar ist. Eine Möglichkeit zur Lösung des Problems wäre ein Compiler, der keinen Maschinencode sondern Java-Quellcode erzeugt, welcher vom Java-Compiler kompiliert und vom Emulator aufgerufen werden kann. Hier liegt ein Problem im Plattform unabhängigen Aufrufen des Compilers. Zudem zwingt dies den Nutzer, zur Nutzung des Programms ein vollständiges Java-Entwicklungssystem bereitzuhalten, was dem Anwender im Interesse der Benutzerfreundlichkeit nicht zugemutet werden sollte. Eine direkte Erzeugung von Java-Bytecode, d.h. die Integration eines Java-Compilers in den Emulator, scheidet aus Komplexitätsgründen aus.

Ich habe mich daher entschlossen, einen eigenen kleinen Compiler und Interpreter zu entwickeln. Beim Laden einer AWL-Datei kompiliert der Compiler den AWL-Quellcode in eine Folge aus Java-Klassenobjekten. Diese Objekte enthalten bereits alle Daten und Parameter in einem vorbereiteten Format, so dass zur Laufzeit des Programms alle Daten bereits vorbereitet sind und das Programm ohne mehrfache Umwandlung direkt von einem Interpreter ausgeführt werden kann.

Die Kommunikation der Programmmodule erfolgt aus den oben genannten Erwägungen zur Latenzminimierung heraus über ein selbst definiertes Unicast-Übertragungsprotokoll, das in einem späteren Kapitel detailliert beschrieben wird. Eine Multicast-Übertragung böte zwar Vorteile bei der Verteilung des SPS-Zustands an die verbundenen Clients, diese Vorteile wiegen jedoch die Nachteile bei Verfügbarkeit und Fehlerkorrektur nicht auf.

3.3. Architektur

Um die Lernplattform möglichst flexibel zu gestalten, wurde sie in vier Hauptmodule unterteilt:

- Emulator-Kern (core)
- Hardware-Interface
- Applikationsmodule
- Steuerkonsole

3.3.1. Emulator-Kern

Der Emulator-Kern (core) übernimmt die Emulation einer SPS. Im Rahmen dieser Diplomarbeit werden nur STEP7-ähnliche Anweisungslisten interpretiert. Weitere Programmiersprachen wie die Funktions- oder Kontaktplandarstellung können später ergänzt werden (intern oder extern durch ein Konvertierungsprogramm).

Der Kern stellt virtuelle Ein- und Ausgänge zur Verfügung, in die nach Wunsch Hardware-Interfaces und Applikationsmodule (auch gemischt) eingeklinkt werden können (vgl. Abbildung „Bild: Modulübersicht“).

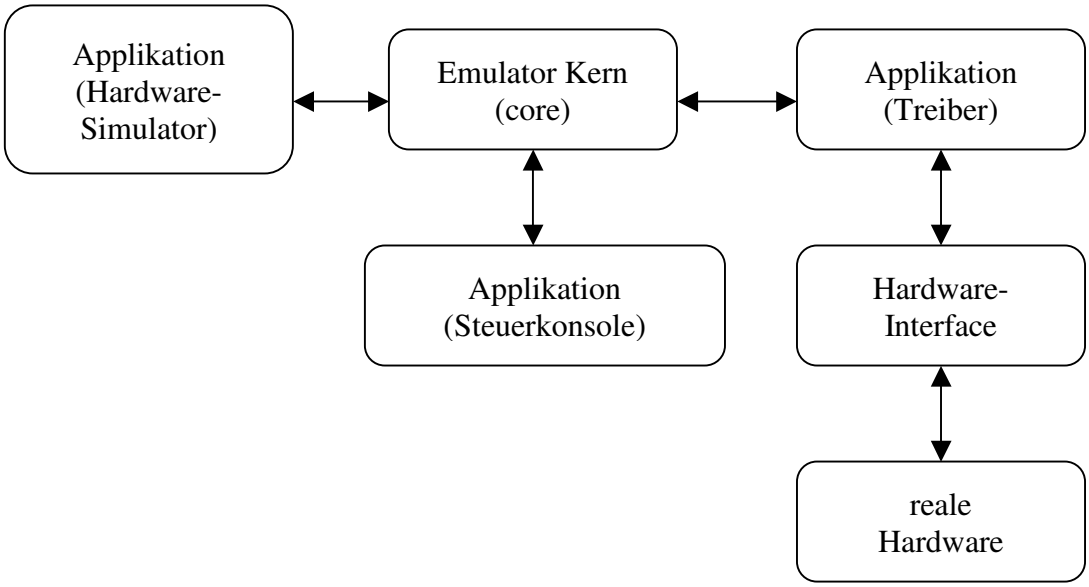


Bild 3.3.1-1: Modulübersicht

Die Module kommunizieren miteinander über ein TCP/IP-Netzwerk, d.h. die Module können je nach Bedarf gemeinsam auf einem PC laufen oder dezentral auf mehrere Rechner innerhalb eines Netzwerkes verteilt werden. Der Emulator-Kern arbeitet dabei als Netzwerkserver, dessen Dienste von den anderen Programmen in Anspruch genommen werden können (vgl. Abbildung „Struktur der Client/Server-Kommunikation“).

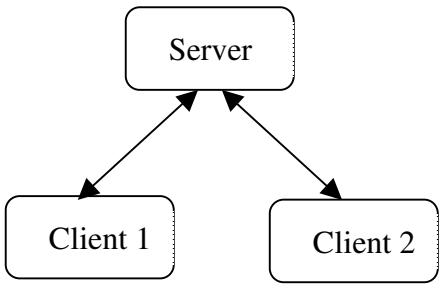
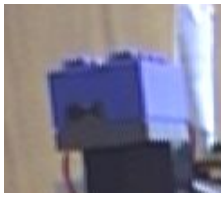


Bild 3.3.1-2: Struktur der Client/Server-Kommunikation

Der Emulator-Kern kommuniziert mit den restlichen Modulen über virtuelle Ein- und Ausgänge, die sowohl einzeln gesteuert als auch auf einer höheren Ebene byte-, wort- oder doppelwortweise angesprochen werden können.
Ein- und Ausgänge sind strikt voneinander getrennt, so dass z. B. der Eingang E0.0 gesetzt sein kann, während der Ausgang A0.0 zurückgesetzt ist. Ebenso ist es möglich, im AWL-Programm, Eingänge zu setzen und Ausgänge auszulesen. Viele kommerzielle SPS-Emulatoren bieten ein Konfigurationsprogramm, mit denen Port-Baugruppen als Ein- oder Ausgänge konfiguriert werden können. Bei physikalischen SPS geschieht diese Auswahl entweder durch den Tausch von Hardwaremodulen (Ein-/Ausgabebaugruppen in dafür vorgesehenen Slots) oder ebenfalls per Softwarekonfiguration.

Dies ist jedoch kein Widerspruch, wie es zunächst scheint, sondern ist durchaus mit dem Verhalten realer Steuerungsgeräte in Einklang zu bringen. Zwar kann eine Anschlussklemme an einer SPS nicht gleichzeitig High- und Lowpegel führen, dennoch sind Ausgänge denkbar, die hochohmig genug sind, um sich von externer Hardware übersteuern zu lassen.



Ebenso ist ein Zeitmultiplexverfahren denkbar, wie es bei der Lego Mindstorms RCX-SPS [Lego01] verwendet wird. Diese verfügt über einen Lichtsensor, der die Umgebung aktiv mit einer LED ausleuchten kann und den reflektierten Lichtwert analog an die RCX-SPS übermittelt. Der Anschluss des Lichtsensors erfolgt über eine Zweidrahtleitung. Das Programm in der Lego-SPS kann die LED im Lichtsensor unabhängig von der Funktion als Sensor für Licht ein- und ausschalten. Es handelt sich also um eine echte bidirektionale Portfunktion.

Da die Eingangs- und Ausgangsbereiche für alle Applikationen gelten, hat der Programmierer der Applikationen dafür Sorge zu tragen, dass sich die einzelnen Applikationen nicht gegenseitig behindern. Dies kann z.B. durch Kommandozeilenparameter geschehen, mit denen der Basisport (auf den sich alle weiteren Ein- und Ausgänge beziehen) gesetzt werden kann.

Der SPS-Emulator der SPS-Lernplattform arbeitet wie die S7-SPS im Zyklusbetrieb. Am Anfang jedes Zyklusses wird ein Prozessabbild erstellt, d.h. alle Eingänge werden gelesen und in den dafür vorgesehenen Speicherbereich des Prozessabbildes übernommen. Das SPS-Programm wird gestartet und liest und beschreibt die Speicherbereiche des Prozessabbildes. Beim Programmende werden alle Änderungen an den Eingängen verworfen (diese können für Testzwecke hilfreich sein, da sie eine interne „Umverdrahtung“ der SPS ermöglichen). Der Zustand der Ausgänge wird mit ihrem Zustand vor dem Programmlauf verglichen und Änderungen werden über das Netzwerk an alle verbundenen Module propagiert.

Parallel wartet der Emulator auf Mitteilungen von Modulen über Änderungen an den SPS-Eingängen. Diese Änderungen können jederzeit empfangen werden, wirksam werden sie jedoch erst mit Beginn des nächsten Zyklusses.

3.3.2. Hardware-Interfaces

Hardware-Interfaces ermöglichen über Treibermodule die Einbindung realer Hardwarekomponenten an die emulierte SPS. Sie setzen die Statusänderungen der emulierten

SPS in Befehle für Schnittstellenkarten um und melden Änderungen an den Eingängen der Hardware an die virtuelle SPS.

3.3.3. Applikationsmodule

Applikationsmodule stellen eine kostengünstige Alternative zur realen Hardware da. Sie simulieren eine kleine Maschine (z.B. einen Aufzug) mit verschiedenen Sensoren und Aktoren, die über die virtuelle SPS programmiert werden können.

3.3.4. Steuerkonsole

Die Steuerkonsole dient zum Steuern des Emulators und zur Fehlersuche. Mit ihr lassen sich Eingänge manuell setzen und Ausgänge auslesen, Programme laden und die Betriebsart (Start/Stop/Wiederanlauf) festlegen. Die Trennung von Konsole und Emulator-Kern ermöglicht auch eine begrenzte Fernsteuerung über das Internet, ohne dass auch die zeitkritische Kommunikation zwischen Kern und Hardware-Interface selbst über das Internet laufen muss. Im Kapitel Anwendungsszenarien werden hierfür Beispiele genannt.

Die SPS-Lernplattform verzichtet auf eine solche Konfiguration und betrachtet alle Ein-/Ausgänge als bidirektional, was gleichzeitig die Bedienung des Programms vereinfacht und die Funktionalität flexibilisiert.

Die Steuerkonsole ermöglicht das Laden von AWL-Programmen in die virtuelle SPS. Die Programme können mit jedem beliebigen Editor (z.B. der Windows Editor, VI oder Emacs) erstellt und mit Hilfe der Steuerkonsole über das Netzwerk in den Emulator geladen werden.

4. Detaillierte Designbeschreibung

4.1. Schnittstelle zwischen den Modulen

Dieses Kapitel beschäftigt sich mit der Kommunikation zwischen dem Server (Emulator-Kern) und den Clients (Konsole, Applikationen).

Als Kommunikationsprotokoll kommt ein von Menschen lesbares ASCII-Protokoll (Klartext) über eine TCP/IP-Verbindung zum Einsatz. Verbindungen werden immer von den Clients aufgebaut und vom Server entgegengenommen.

4.1.1. Allgemeiner Aufbau der Protokollbefehle

Alle Befehle vom oder zum Server werden als sogenannte Protokollzeilen gesendet. Eine Protokollzeile beginnt immer mit einem Schlüsselwort (Befehl). Optional können dem Schlüsselwort weitere Parameter folgen. Abgeschlossen wird jede Zeile durch einen Zeilenumbruch („\n“).

Befehl	Parameter1... Parameter *n*	\n

Da die zugrunde liegende TCP/IP-Verbindung für Fehlerkorrektur während der Übertragung sorgt, werden Befehle nicht quittiert. Eine Beantwortung erfolgt ebenfalls nicht. Allerdings kann das Senden eines Befehls den Empfänger veranlassen, als Reaktion darauf selbst bestimmte Befehle zu senden. Dies kann als indirekte Antwort auf den Ursprungsbefehl betrachtet werden, erfolgt jedoch asynchron, da das Senden und Empfangen unabhängig voneinander funktioniert.

Zur Fehlersuche kann es hilfreich sein, mit „telnet *servername serverport*“ eine Verbindung zum Server aufzubauen und die Datenübertragung auf Protokollzeilenebene zu verfolgen. Zur Simulation eines Servers kann das Programm netcat verwendet werden. (netcat –l –p *serverport*)

4.1.2. Befehle vom Client zum Server

4.1.2.1. Verfügbare Befehle

In der aktuellen Programmversion können die Clients die folgenden Befehle an den Server senden:

Befehl:	Parameter:	Beschreibung:
: (Doppelpunkt)	Codezeile	Überträgt AWL-Quellcode zum Server, siehe „Hochladen von Code“.
compile	Optional: Bausteinname	Kompiliert den Code im Upload-Puffer. Als Parameter kann ein Bausteinname angegeben werden, unter dem der Code abgelegt werden soll (z.B. FB1). Wird kein Bausteinname angeben, versucht der Server den Namen aus einer Direktive innerhalb des Quellcodes zu ermitteln. Siehe „Hochladen von Code“.

down	-	Fährt den Server herunter.
dump	-	Gibt den Inhalt der worldmap aus. (Debugging)
dumpi	-	Gibt den Inhalt der cyclemap aus. (Debugging)
quit	-	Beendet die Verbindung zum Client.
runmode	„cold“, „warm“ oder „stop“	Setzt die Betriebsart der SPS: cold: Kaltstart warm: Warmstart stop: Inaktiv Beispiel: „runmode cold“
set	*L*:*V*:*M [L*:*V*:*M]*...	Setzt SPS-Eingänge: *L*: Nummer des Byteports (hexadezimal) *V*: Wert des Byteports (hexadezimal) *M*: Bitmaske mit den gültigen Bits in V. (hexadezimal) Siehe „Setzen von Eingängen“.
upload	-	Löscht den Upload-Puffer und bereitet eine neue Übertragung vor. Siehe „Hochladen von Code“.

4.1.2.2. Hochladen von AWL-Quellcode

Das Hochladen von AWL-Quellcode vom Client auf den Server geschieht in drei Schritten:

1. Der Client sendet den Befehl „upload“, um den Server über eine bevorstehende Übertragung zu informieren. Damit werden die Übertragungspuffer zurückgesetzt.
2. Der Client sendet den Quellcode des Programms. Jeder Zeile wird dabei ein Doppelpunkt („:“) vorangestellt, um eine Fehlinterpretation des Quellcodes als Netzwerkbefehl zu verhindern.
3. Der Client schließt die Übertragung mit dem Befehl „compile“ ab. Die Möglichkeit, den Bausteinnamen manuell vorzugeben, wird vom Client derzeit nicht verwendet.

Beispielübertragung
upload : ORGANIZATION_BLOCK OB1 : : L EW0 : T AW0 : : INVI : : T AW2 : : BE compile

4.1.2.3. Setzen von Eingängen

Über den set-Befehl kann ein Client virtuelle Eingänge am SPS-Modell im Server setzen. Die Eingänge sind byteweise organisiert. Intern werden die Eingänge in einem Array aus 32-Bit-

Integern gespeichert, von denen nur die untersten 8 Bit benutzt werden. (Die Anzahl der Eingänge steht in der Konstanten shared.Const.MAXIOBYTES im Java-Programm).

Dem set-Befehl folgen ein oder mehrere Tupel im Format *L*:*V*:*M*

L, *V* und *M* sind Zahlen im Hexadezimalsystem.

L bezeichnet die Nummer des Byteeingangs (0 bis MAXIOBYTES-1). *V* enthält den Wert, auf den der Byteport gesetzt werden soll. Dies betrifft jedoch nur die Bits, die in der Maske *M* gesetzt sind. Alle Eingänge, deren Bit in der Maske auf 0 steht, bleiben unverändert.

Beispiele:

Befehl:	Bedeutung:
set 0:1:1	setzt E0.0
set 0:2:2	setzt E0.1
set 0:0:1	setzt E0.0 zurück
set: 7:10:10	setzt E7.4
set 0:1:11	E0.0 setzen und E0.4 rücksetzen
set 0:15:FF	setzt Byteport EB0 auf 0x15
set 0:15:FF 7:10:10	setzt Byteport EB0 auf 0x15, setzt E7.4

Ein Client sollte so strukturiert sein, dass die Eingänge mit möglichst wenigen Operationen (möglichst wenige set-Befehle mit möglichst wenigen Parametern) gesetzt werden, um die Latenzzeiten und das Datenaufkommen gering zu halten. Beim Programmstart sollten alle benutzten Eingänge initialisiert werden.

Die Java-Klasse AppFramework bietet bereits fertige Funktionen zur Kommunikation mit dem Server und sollte daher als Basis für eigene Applikationen benutzt werden.

4.1.3. Befehle vom Server zum Client

4.1.3.1. Verfügbare Befehle

Der Client wird vom Server durch die folgenden Befehle über Statusänderungen informiert:

Befehl:	Parameter:	Beschreibung:
update	*L*=*V* [*L*=*V*] ...	Informiert den Client, dass sich der Wert des Ausgangsbytes *L* auf den Wert *V* geändert hat. Beide Zahlen werden hexadezimal angegeben. Beispiel: „update 0=FF 1=0 C=0 D=F1 E=0 F=80“
text	ASCII-Text	Der text-Befehl informiert den Benutzer über das Ergebnis von Befehlen, die Version des Servers, Fehler, Warnungen und andere wichtige Zustandsdaten. Der Client gibt den empfangenen Text aus. Die Klasse AppFramework benutzt standardmäßig die Java-Standardausgabe (stdout). Durch Überschreiben der Methode toCons kann die Ausgabe jedoch auch

		umgeleitet werden, z.B. in ein Textfenster. Beispiel: „text SPS Emulator 0.27 by Lars Struss <lstruss@gmx.net> 2005“
runmode	„cold“, „warm“ oder „stop“	Informiert den Client über Änderungen der Betriebsart der SPS: cold: Kaltstart warm: Warmstart stop: Inaktiv Beispiel: „runmode stop“

4.1.3.2. Statusänderungen an Ausgängen

Der Server informiert die Clients beim Verbindungsaufbau und nach jedem Zyklus über Änderungen an den SPS-Ausgängen. Die Ausgänge sind ebenso wie die Eingänge byteweise organisiert. Intern werden die Ausgänge in einem Array aus 32-Bit-Integern gespeichert, von denen nur die untersten 8 Bit benutzt werden. (Die Anzahl der Ausgänge steht in der Konstanten shared.Const.MAXIOBYTES im Java-Programm).

Beim Verbindungsaufbau sendet der Server einmalig einen update-Befehl für alle Ausgänge an den Client, um einen gemeinsamen Datenbestand herzustellen.

Nach jedem Programmzyklus vergleicht der Server den Zustand der Ausgänge mit dem Zustand der Ausgänge vor dem Programmzyklus. Wird hierbei mindestens eine Änderung festgestellt, so sendet der Server einen update-Befehl mit allen Änderungen. Auf die Übertragung des vollständigen Zustandsvektors kann verzichtet werden, da die Fehlerkorrektur bereits auf TCP/IP-Ebene durchgeführt wird und deswegen kein Datenverlust zu erwarten ist.

Die Updates übertragen immer komplette Bytes. Daher ist eine Bitmaske wie beim set-Befehl nicht erforderlich. Es obliegt der Applikation, sich die benötigten Bits herauszufiltern.

4.2. Struktur der Software

Dieses Kapitel erklärt die Zusammenhänge innerhalb des Programms und richtet sich an Entwickler, die die Software erweitern wollen.

4.2.1. Java-Packages

Zur einfacheren Handhabung wurde die Software in folgende Java-Packages unterteilt:

Package:	Inhalt:
apps	Bibliotheksfunktionen für Applikationen
apps.console	Steuerkonsole
apps.elevator	Fahrstuhl-Beispiel-Applikation
core	Emulator-Kern
shared	gemeinsame Komponenten und projektweite Konstanten

Eine genaue Übersicht über alle Klassen findet sich in der JavaDoc-Dokumentation im Anhang dieses Dokumentes. Auf UML-Diagramme wurde verzichtet, da sie viel Platz benötigen und keinen über die JavaDoc-Dokumentation hinausgehenden Inhalt besitzen.

4.2.2. Übersicht über den Emulator-Kern

Der Kern unterteilt sich in die folgenden Module (die Pfeile beschreiben den Datenfluss zwischen den Modulen):

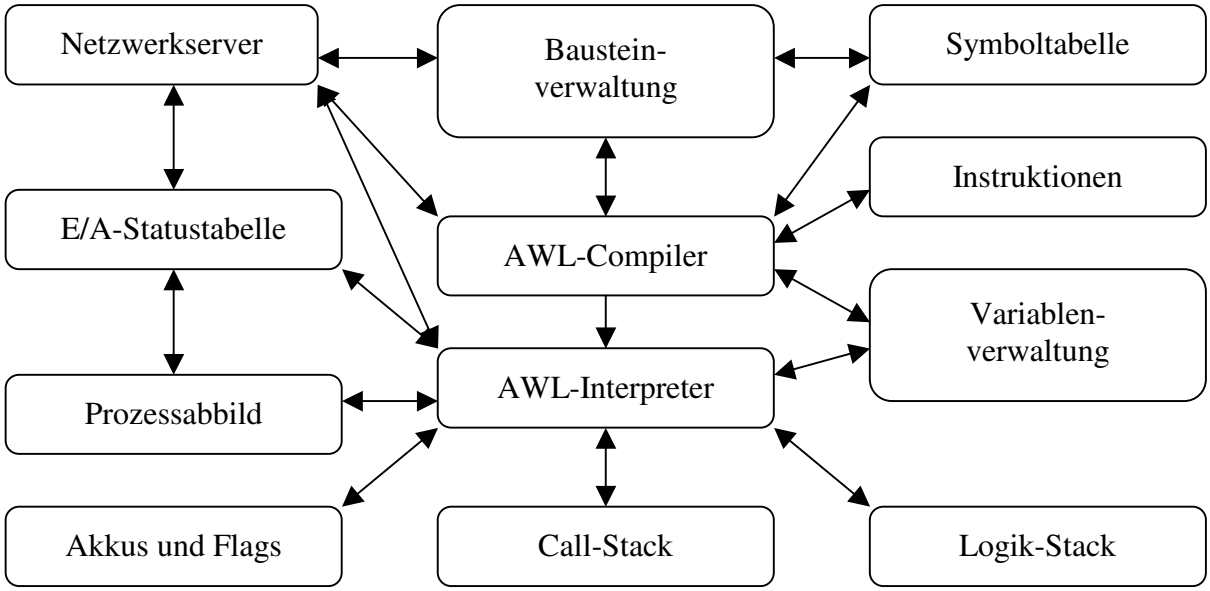


Bild 4.2.2-1: Übersicht über den Emulator-Kern

Modul:	Funktion:
Netzwerkserver	Annahme und Verwaltung von eingehenden Netzwerkverbindungen und Steuerung der anderen Module auf Basis der empfangenen Befehle.
E/A-Statustabelle	Enthält den Status aller Ein- und Ausgänge der SPS. Die Eingänge werden dabei über Netzwerkbefehle von den Applikationen gesetzt, während die Ausgänge nach jedem Programmzyklus aus dem Prozessabbild übernommen werden.
Prozessabbild	Enthält eine Kopie der Zustände aller Ein- und Ausgänge der SPS. Auf dem Prozessabbild erfolgt die Abarbeitung des laufenden Programms. Die Eingänge werden am Anfang jedes Programmzyklusses aus der E/A-Statustabelle übernommen. Am Ende des Zyklusses werden die Ausgänge in die E/A-Statustabelle geschrieben.
Akkus und Flags	Enthält Zwischenergebnisse von Berechnungen.
Bausteinverwaltung	Speichert und verwaltet SPS-Programm- und Datenbausteine.
AWL-Compiler	Kompiliert den AWL-Quellcode eines Bausteins in Java-Klasseninstanzen.
AWL-Interpreter	Führt das kompilierte AWL-Programm (die Gesamtheit aller Bausteine) aus.
Call-Stack	Speichert und verwaltet Rücksprungadressen bei verschachtelten Bausteinaufrufen.
Symboltabelle	Speichert Parameter und lokale Variablen.
Instruktionen	Eine Gruppe von Klassen, die die Funktion der einzelnen AWL-Instruktionen implementiert.
Variablenverwaltung	Speichert globale und statische Variablen.
Logik-Stack	Speichert Zwischenergebnisse bei verschachtelten Logikoperationen.

4.3. Entwicklung zusätzlicher Applikationen mit der Package apps

4.3.1. Funktionsübersicht über die Package apps

Die Package apps stellt die zentrale Klasse AppFramework bereit, von der eigene Klassen abgeleitet werden können. Neue Applikationen sollten auf diese Klasse aufsetzen, da sie eine Abstraktionsebene zwischen Server und Anwendung bietet und die Anwendung dadurch besser mit zukünftigen Serverversionen zusammenarbeiten kann.

Die Klasse kümmert sich um die folgenden Funktionen:

- Empfang von Statusänderungen an SPS-Ausgängen und Aufbereitung der Daten für die Applikationsklasse
- Setzen und Rücksetzen von SPS-Eingängen, automatische Erzeugung von inkrementellen Updates und vollständige Übertragung aller Daten bei einem Verbindungsneuaufbau.
- Pufferung der Daten (Ein- und Ausgänge), damit die aktuellsten Informationen jederzeit verfügbar sind.
- Aufbereitung der Änderungen der Betriebszustände der SPS.
- Initialisierung eines Programmfensters zur Interaktion mit dem Benutzer
- Vereinfachung der Java-Menüprogrammierung durch Hilfsfunktionen zum einfachen Erstellen von Menüpunkten mit Menüverwaltung zum einfachen Aktivieren und Deaktivieren von mehreren Menüeinträgen
- Funktionen zum Hochladen von AWL-Programmen in die SPS, optional auch mit fertiger Dateiauswahl-Dialogbox und Benutzerführung
- Integrierte Textkonsole zum Empfang von Servertextmeldungen mit Möglichkeit zur Umleitung der Daten (z.B. in Fenster oder Dateien)

Diese Funktionalität wird durch Methoden, Eventhandler und Klassenattribute realisiert.

Wichtige Methoden der Klasse AppFramework:

Methode:	Funktion:
void **connect**(String nethost, int netport)	Verbindung mit dem Server herstellen
void **disconnect**()	Verbindung zum Server trennen
protected void **enableMenus**(String menuList, boolean enabledFlag)	Aktiviert oder deaktiviert mehrere Menüs.
protected JcheckBoxMenuItem **JCheckBoxMenuItem**(JMenu mnu, String key, String text, int mnemonic)	JCheckBoxMenuItem-Menüeintrag erzeugen und in menuTable eintragen
protected JMenuItem **JMenuItem**(JMenu mnu, String key, String text, int mnemonic)	JMenuItem-Menüeintrag erzeugen und in menuTable eintragen
protected JRadioButtonMenuItem **JRadioButtonMenuItem**(JMenu mnu, String key, ButtonGroup bgrp, String text, int mnemonic, boolean selected)	JRadioButtonMenuItem-Menüeintrag erzeugen und in menuTable eintragen
void **quit**(boolean shutdown)	Appplikation beenden
void **sendLine**(String cmd)	Protokollzeile an den Server senden

protected void **showConnectDialog**()	Verbindungsdialog anzeigen Die Methode öffnet eine Dialogbox, in der der Benutzer Serveradresse und Port eintragen kann.
protected void **toCons**(String line)	Ausgabe von Statusmeldungen (Konsole)
boolean **update**(int location, int value, int mask)	SPS-Eingang setzen
int **updateAll**()	Alle SPS-Eingänge an den Server übertragen.
void **uploadAWL**()	Upload-Dialogbox für AWL-Programme und Batch-Listen öffnen
void **uploadAWL**(File file, String blockname)	AWL-Anweisungsliste an den Server senden

Wichtige Eventhandler der Klasse AppFramework:

Methode:	Funktion:
protected void **notifyOffline**()	Diese Methode wird beim Trennen der Serververbindung aufgerufen.
protected void **notifyOnline**()	Diese Methode wird beim Verbindungsaufbau zum Server aufgerufen.
protected void **notifyOutputUpdate**(int location, int value, int changedMask)	Diese Methode wird bei Pegeländerungen an den SPS-Ausgängen aufgerufen.
protected void **notifyRunModeChange**(int runmode)	Diese Methode wird bei Änderungen der Betriebsart der SPS-CPU aufgerufen.
protected void **toCons**(String line)	Ausgabe von Statusmeldungen (Konsole)

Wichtige Attribute der Klasse AppFramework:

Methode:	Funktion:
protected File **currentDirectory**	Referenz auf das zuletzt verwendete Verzeichnis
protected int[] **inMasks**	Bitmasken zum inValues-Array
protected int[] **inValues**	Array mit den aktuellen Werten der SPS-Eingänge.
protected JFrame **mainWindow**	Vordefiniertes Hauptfenster
protected Hashtable<String,JMenuItem> **menuTable**	Menütabelle Hier werden alle Menüs des Hauptfensters gespeichert um sie über symbolische Namen einfach ansprechen zu können.
protected String **nethost**	DNS-Name oder IP-Adresse des Servers
protected int **netport**	TCP/IP-Portnummer des Servers
protected NetSocket **nsock**	Referenz auf eine mit dem Server verbundene NetSocket-Instanz oder null, falls keine Verbindung besteht
protected int[] **outValues**	Array mit den aktuellen Werten der SPS-Ausgänge

Wichtige Konstanten der Klasse AppFramework:

Methode:	Funktion:
static int **RM_COLD**	Konstante für die Betriebart „Kaltstart“ der SPS

static int **RM_WARM**	Konstante für die Betriebart „Warmstart/Wiederanlauf“der SPS
static int **RM_STOP**	Konstante für die Betriebart „Stop“ der SPS

4.3.2. Treiber für Hardware-Interfaces

Zur Entwicklung von Treibern für Hardware-Interfaces für die SPS-Lernplattform gibt es zwei grundsätzliche Methoden:

a) Entwicklung einer Java-Applikation basierend auf der Klasse AppFramework mit Hardwarezugriffen über Java-API-Klassen oder das *Java Native Interface* (JNI, [Sun02]).
b) Implementierung des Netzwerkprotokolls in einer systemnahen Programmiersprache wie z.B. C oder C++.

Aufgrund der vielfältigen Hardwareschnittstellen gibt es keine Universallösung, welche allen Problemstellungen gerecht wird. Die richtige Strategie hängt daher von der zu programmierenden Hardware ab.

4.3.2.1. Treiber für Hardware-Interfaces an Standardports

Für Interfaces zum Anschluss an den seriellen Port (z.B. Festo EasyPort oder Lego Mindstorms RCX) empfiehlt sich die Entwicklung einer kleinen Java-Applikation, welche die Klasse AppFramework zur Kommunikation mit dem Server und das *Java Communications API* [Sun01] zur Kommunikation mit dem Hardwareinterface einsetzt. Die Plattformunabhängigkeit bleibt dabei vollständig erhalten.

4.3.2.2. Treiber für Hardware-Interfaces an speziellen Ports

Für Interfaces, die nicht über ein vorhandenes Java-API angesprochen werden können, muss eine plattformspezifische Zugriffsmethode gewählt werden. Die beste Möglichkeit besteht hier darin, alle plattformabhängigen Operationen in eine externe Bibliothek auszulagern, welche in einer systemnahen Sprache wie C oder C++ geschrieben ist und über das Java Native Interface aufgerufen werden kann.

4.3.2.3. Ausgeben von elektrischen Signalen

Der Eventhandler notifyOutputUpdate(...) wird von AppFramework aufgerufen, sobald der Emulator-Server Änderungen an den SPS-Ausgängen bekannt gibt. Durch Überlagern dieser Methode in der Applikationsklasse kann eine JNI-Methode zum Setzen von Ausgängen am Hardwareinterface aufgerufen werden.

4.3.2.4. Einlesen von elektrischen Signalen

4.3.2.4.1. Zyklische Abfrage (Polling)

Wenn das Interface keine Signalisierung von Signaländerungen durch Eventhandler, Signale oder Interrupts unterstützt, muss das Interface in regelmäßigen Zeitabständen abgefragt („gepollt“) werden. Hierbei muss ein Kompromiss zwischen CPU-Belastung und

Reaktionszeit gefunden werden. Außerdem besteht die Gefahr, dass kurze Impulse am Eingang übersehen werden, wenn die Impulsdauer kürzer als die Zeit zwischen zwei Abfragen ist. Zur Meldung der Änderungen an den Server kann die update(...)-Methode der Klasse AppFramework verwendet werden. Sie prüft automatisch, ob eine Änderung vorliegt, und erzeugt bei Bedarf eine passende Protokollzeile für den Server.

4.3.2.4.2. Eventbasiert (Interrupt)

Wenn das Interface eine Signalisierung von Signaländerungen durch Eventhandler, Signale oder Interrupts unterstützt, können Änderungen an den Interface-Eingängen bei Benachrichtigung durch die Hardware verarbeitet und durch Aufruf der update(...)-Methode an den Server gemeldet werden. Dieses Verfahren bietet eine schnelle Reaktion auf Pegeländerungen bei minimaler CPU-Beanspruchung.

Da die Möglichkeiten, in Java auf Hardwareinterrupts zu reagieren, auch bei Verwendung des JNI sehr eingeschränkt sind, kann es bei komplizierterer Hardware sinnvoll oder notwendig sein, auf den Java-Unterbau zu verzichten und den kompletten Treiber in einer systemnahen Programmiersprache wie C oder C++ zu schreiben. Hierbei können alle Vorteile der jeweiligen Hardwareplattform ausgenutzt werden. Allerdings muss auch das Netzwerkprotokoll zur Kommunikation mit dem Server selbst implementiert werden. Eine Dokumentation aller Netzwerkbefehle und des Protokollaufbaus befindet sich im diesbezüglichen Kapitel dieser Anleitung.

4.3.2.5. Brücken-Applikationen

Applikationen können auch als Brücken zu anderen Systemen funktionieren, z.B. zu anderen Hardware-Simulatoren. Hierfür muss eine Applikation programmiert werden, die die Statusänderungen an allen relevanten Ein- und Ausgängen in das Übertragungsprotokoll des jeweiligen Zielsystems umwandelt. Durch den reichhaltigen Vorrat des Java-Systems an Netzwerkfunktionen sollte sich die SPS-Lernplattform so in die meisten Systeme einbinden lassen.

4.3.2.6. Andere Hardwareplattformen

Selbstverständlich kann der Treiber auch auf einem Mikrokontroller oder einer (realen) SPS ausgeführt werden, wenn die Hardware eine Netzwerkschnittstelle besitzt oder ein entsprechender Medienkonverter vorhanden ist.

4.3.2.7. Fail-Safe-Funktion

Da der Treiber über eine TCP/IP-Verbindung an den Server angebunden ist, sollte schon bei der Entwicklung des Treibers bedacht werden, dass Netzwerkverbindungen bei Netzwerkproblemen (z.B. Routerausfall oder Überlastung einzelner Netzrouten) oder Softwareproblemen zusammenbrechen können. Außerdem kann es bei TCP/IP zuweilen zu langen Timeouts bei Paketverlusten kommen, die das Zeitverhalten zwischen Emulator-Server und Applikationen beeinträchtigen können. Treiber für Hardwareinterfaces sollten daher bei aufwendigeren Modellen unbedingt eine Failsafe-Funktion enthalten, die das Modell bei einem Verbindungsabbruch in einen sicheren Zustand bringt und den Aufbau einer neuen Verbindung versucht. Die Applikation kann darüber hinaus in regelmäßigen Abständen

einen undefinierten Befehl wie z.B. „PING!“ senden und über die daraus resultierende Fehlermeldung überprüfen, ob der Server noch erreichbar ist.

Da auch der Treiber abstürzen kann, sind evtl. auch zusätzlich Hardwareschutzschaltungen über die gesetzlichen Bestimmungen hinaus (z.B. Not-Aus, Endschalter) sinnvoll.

4.4. Erweiterung des Emulators (Servers) um weitere Befehle

4.4.1. Umsetzung des Emulators

Bereits bei der Entwicklung des Servers wurde auf leichte Erweiterbarkeit geachtet.

Dreh- und Angelpunkt der Instruktionsverwaltung ist die Klasse *Instruction*. Im Array *opcodeList[]* wird allen Instruktionen der SPS eine Klasse zugeordnet:

Instruction.opcodeList[]

```
/** Eine Liste mit allen unterstützten Opcodes im Format
"opcode;package.klassenname". */
 protected static final String[] opcodeList={
  "ORGANIZATION_BLOCK;",
  "FUNCTION_BLOCK;",
  "DATA_BLOCK;",
  "U;core.InstrLogic1",
  "UN;core.InstrLogic1",
  "O;core.InstrLogic1",
  "ON;core.InstrLogic1",
  "X;core.InstrLogic1",
[...]
```

Rein deklarative Anweisungen wie „`ORGANIZATION_BLOCK`“ haben keine Klassenangabe, da diese Anweisungen nur die Kompilierung steuern und nicht zur Laufzeit ausführbar sind.

Wenn der Baustein-Compiler eine AWL-Anweisung zu kompilieren beabsichtigt, ruft er dazu die statische Methode *create()* aus der Klasse *Instruction* auf. Diese ermittelt die zuständige Instruktionsklasse und instantiiert sie. Aufgrund dieses dynamischen Ladevorgangs enthält der Quellcode keinen direkten (Code-)Bezug zu den Instruktionsklassen. Sie können also vom Java-Compiler nicht automatisch erkannt und mitkompiliert werden, sondern müssen manuell mit javac in eine entsprechende class-Datei übersetzt werden.

Nach der Instantiierung fährt *create()* mit der Initialisierung der Klasse fort.
Alle Instruktionsklassen sind von der abstrakten Klasse *Instruction* abgeleitet, die die grundlegende Funktionalität (z.B. Zugriff auf diverse CPU-Bereiche, Speicherung von Opcode- und Bausteinnamen sowie die Zeilennummer im AWL-Quelltext) für alle Befehle bereitstellt.

4.4.2. Erweiterung des Befehlsvorrats

Jede Instruktionsklassen muss

- von *Instruction* abgeleitet sein,
- eine Methode `protected boolean parseParam(AWLParser parser)` definieren und

- eine Methode `public boolean execute()` implementieren.

Die Methode *parseParam()* wird während der Kompilierung aufgerufen und bekommt eine Instanz der Klasse *AWLParser* übergeben, über die sie die Parameter der Instruktion ermitteln kann. Wenn die Instruktionsklasse dadurch alle nötigen Parameter auslesen konnte, muss sie mit dem Wert *true* zum Aufrufer zurückkehren. Fehlten Parameter oder hatten Parameter die falsche Reihenfolge oder einen falschen Datentyp (z.B. Label anstatt Byteport), so sollte sie *false* zurückliefern. Überschüssige Parameter können ignoriert werden, diese werden vom Compiler erkannt und automatisch als Syntaxfehler gemeldet.

Die Methode *execute()* wird zur Laufzeit aufgerufen, sobald die Programmausführung die Instruktion erreicht. Sie aktualisiert den CPU-Zustand entsprechend der Semantik des Opcodes und liefert bei Erfolg *true* zurück. Tritt ein Laufzeitfehler auf, muss die Methode als Funktionswert *false* liefern, der Emulator beendet dann die Programmausführung mit einer Fehlermeldung und schaltet die SPS in die Betriebsart „Stop“.

4.5. Grenzen der SPS-Lernplattform

Selbstverständlich kann und soll dieser SPS-Emulator die Siemens SIMAC-Software nicht 1:1 nachbilden, denn dies würde nicht nur rechtliche Probleme aufwerfen, sondern wäre auch im Rahmen einer Diplomarbeit nicht zu schaffen.

Das Programm unterliegt in der aktuellen Version den folgenden Einschränkungen:

- Das Programm enthält keine Benutzerverwaltung. Alle Befehle, auch die Befehle zum Beenden des Servers, können von jedem Benutzer ausgeführt werden. Wenn das Programm ständig verfügbar sein soll, so empfiehlt sich ein Startskript, das den Server nach dem Beenden oder bei Abstürzen automatisch neu startet. Außerdem kann der Benutzerkreis, dem der Server zugänglich ist, mit Hilfe einer Firewall und Netzwerk-Tunneln/Port-Forwarding (z.B. über ssh) eingeschränkt und kontrolliert werden.
- Es wird nicht zwischen verschiedenen Startarten wie Kaltstart und Wiederanlauf unterschieden.
- Nur OB100 wird aufgerufen. Alle anderen OB können zwar geladen werden, sind jedoch ohne Funktion.
- Es gibt keinen integrierten AWL-Quellcodeeditor, hier kann jedoch jeder beliebige Texteditor verwendet werden.
- Es gibt keinen Symboltabelleneditor. Alle Variablen müssen textuell im Quellcode definiert werden.
- Der Emulator macht keinen Unterschied zwischen Bausteinparametern vom Typ INPUT und OUTPUT. Alle Parameter werden als Referenz übergeben und können vom aufgerufenen Baustein verändert werden.
- Beim Übergang in die Betriebsart „Stop“ bleiben alle Ausgangszustände erhalten. Eventuell angeschlossene Hardware sollte daher über eine eigene Not-Aus-Funktion verfügen oder eine Failsafe-Routine im Interfacetreiber implementieren, da es jederzeit zum Abriss der Verbindung zum Emulator kommen kann. (z.B. durch fremden Netzwerkverkehr, der das Netz überlastet)
- Es gibt bisher keine Systembibliothek (SFC-Aufrufe)
- Das Master-Control-Relais (MCR) wird nicht unterstützt.

Achtung: Diese Software dient dem Einsatz in der Lehre. Sie ist nicht für den produktiven Betrieb ausgelegt. Auf keinen Fall darf sie in Maschinen verwendet werden, in denen eine Fehlfunktion Menschen oder Tiere verletzen oder töten kann. Die Verwendung dieser Software erfolgt auf eigenes Risiko!

4.6. Testmethodik

4.6.1. Komponententests

Der Test der Software erfolgte auf mehreren Ebenen. Bereits während der Entwicklung wurden einzelne Programmteile durch Testaufrufe getestet. Das Programm wurde zudem um Selbsttestfunktionen und Plausibilitätsprüfungen erweitert, die Fehlfunktionen melden und so verhindern, dass Fehler in internen Datenstrukturen unerkannt bleiben. Einige Plausibilitätsprüfungen wurden aus Performancegründen in der aktuellen Version der Software wieder auskommentiert, können jedoch bei Bedarf wieder im Quellcode aktiviert werden.

4.6.2. Systemtests

Zur Prüfung des Gesamtsystems wurden unterschiedliche AWL-Programme auf dem Emulator gestartet und die Ergebnisse des Programms mit dem Verhalten des AWL-Programms auf anderen Emulatoren (z.B. WinSPS-S7) verglichen.

Eine Sammlung mit Testprogrammen befindet sich auf der beiliegenden CD und ist im Anwenderhandbuch detailliert beschrieben.

5. Anwenderhandbuch

Dieses Kapitel beschäftigt sich mit der SPS-Lernplattform aus Sicht des Anwenders und beschreibt die Installation, das Starten der Software und die ersten Schritte im Programm.

5.1. Installation

5.1.1. Systemanforderungen:

Der Emulator benötigt ein installiertes Java-Runtime-Enviroment (JRE) oder Java-Development-Kit (JDK) der Version 1.5.0 oder höher. Eine Installation auf einem älteren JRE/JDK wird fehlschlagen, da die Software auf neue Compilerfunktionen und Bibliotheken zurückgreift.

Eine Liste der unterstützten Systemkonfigurationen befindet sich unter http://java.sun.com/j2se/1.5.0/system-configurations.html .

5.1.2. Installation der Java-Laufzeitumgebung

Die Java-Laufzeitumgebung gibt es in zwei Paketen. Zur Benutzung dieser Software reicht das kleinere JRE- Paket. Das größere JDK ist nur für Entwickler wichtig, die die Software verändern und/oder erweitern möchten. An dieser Stelle beschränke ich mich daher auf die Beschreibung der Installation des JRE.

1. Falls noch nicht geschehen (z.B. für die Java-Darstellung im Webbrowser), muss zunächst das JRE, wie unter http://java.sun.com/j2se/1.5.0/jre/install.html und http://java.sun.com/j2se/1.5.0/download.jsp beschrieben, heruntergeladen und installiert werden.

2. Unter Windows sollte das Java-Verzeichnis in den Suchpfad aufgenommen werden, um nicht bei allen Java-Befehlen den kompletten Pfad zur Datei „java.exe“ eingeben zu müssen. Dies geschieht unter Windows 2000/XP, in dem in der Windows Systemsteuerung unter „System“ → „Erweitert“ → „Umgebungsvariablen“ die Variable „PATH“ modifiziert wird (siehe Abbildung „Windows-Suchpfad“). Eine genaue Anleitung für alle wichtigen Windowssysteme findet sich unter http://java.sun.com/j2se/1.5.0/install-windows.html

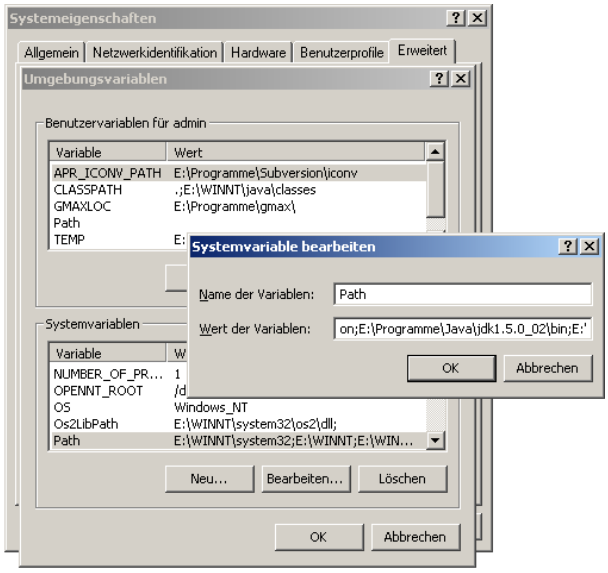


Bild 5.1.2-1: Windows-Suchpfad

Damit ist die Installation der Java-Laufzeitumgebung abgeschlossen. Die Installation war erfolgreich, wenn die Eingabe von „java“ in einer Eingabeaufforderung (Windows-Eingabeaufforderung oder Shell unter Unix) zur Ausgabe eines Hilfetextes führt. Erscheint lediglich eine Meldung, dass der Befehl unbekannt ist oder nicht gefunden wurde, so ist die Installation oder der Suchpfad fehlerhaft.

5.1.3. Installation des Emulators und der Anwendungen

Zur Installation des Emulators und der Beispielanwendungen reicht es, das Programm von der Installations-CD in ein leeres Unterverzeichnis auf der Festplatte zu kopieren bzw. das heruntergeladene Archiv zu entpacken.

5.2. Starten des Programms

5.2.1. Starten des Emulators

Der Emulator (Server) wird durch Doppelklick auf das Startskript ***emu.bat*** gestartet.

Alternativ kann das Programm auch manuell gestartet werden. Beispiel: Der Emulator wurde unter Windows im Verzeichnis c:\spsemu installiert:

Windows
c: cd \spsemu java -jar emu.jar

Linux
cd /spsemu java -jar emu.jar

Beim manuellen Start kann das Programmverhalten durch zusätzliche Kommandozeilenparameter am Ende der Zeile beeinflusst werden:

Parameter:	Funktion:
-a *dateiname.awl*	Lädt die angegebene Datei als Baustein in die SPS. Dieser Parameter ist hilfreich, wenn einige Bausteine (z.B. Bibliotheken) automatisch beim Programmstart geladen werden sollen. Der Parameter kann mehrfach angegeben werden, wenn mehrere Dateien geladen werden sollen.
-p *portnummer*	Ändert die TCP/IP-Portnummer, auf der der Emulator auf Verbindungen von Clients (z.B. Konsole oder Applikationen) wartet. Die Portnummer muss im Clientprogramm eingeben werden. Wenn der Parameter nicht angegeben wird, wird Port 5000 benutzt. Die Angabe dieser Option ist daher nur notwendig, wenn der Port 5000 bereits von einem anderen Programm verwendet wird oder mehrere SPS-Emulatoren auf einem Computer laufen sollen. Achtung: Für Portnummern kleiner als 1024 sind auf manchen Betriebssystemen Adminstrator/root-Rechte erforderlich.
-s *nanosekunden*	Setzt die Verzögerungszeit nach jedem Programmzyklus. Wenn der Wert größer als 0 ist, übergibt der Emulator nach jedem AWL-Programmzyklus die Kontrolle für die angegebene Zeit an das Betriebssystem. Dies ermöglicht eine Anpassung der SPS-Geschwindigkeit an die Anforderungen der Simulation und verhindert, dass Rechenzeit verschwendet wird. Dadurch reduziert sich auch der Stromverbrauch und die Wärmeentwicklung der CPU.

Beispiel: java -jar emu.jar –p 5010

Das Programm meldet sich mit einer Ausgabe ähnlich dieser:

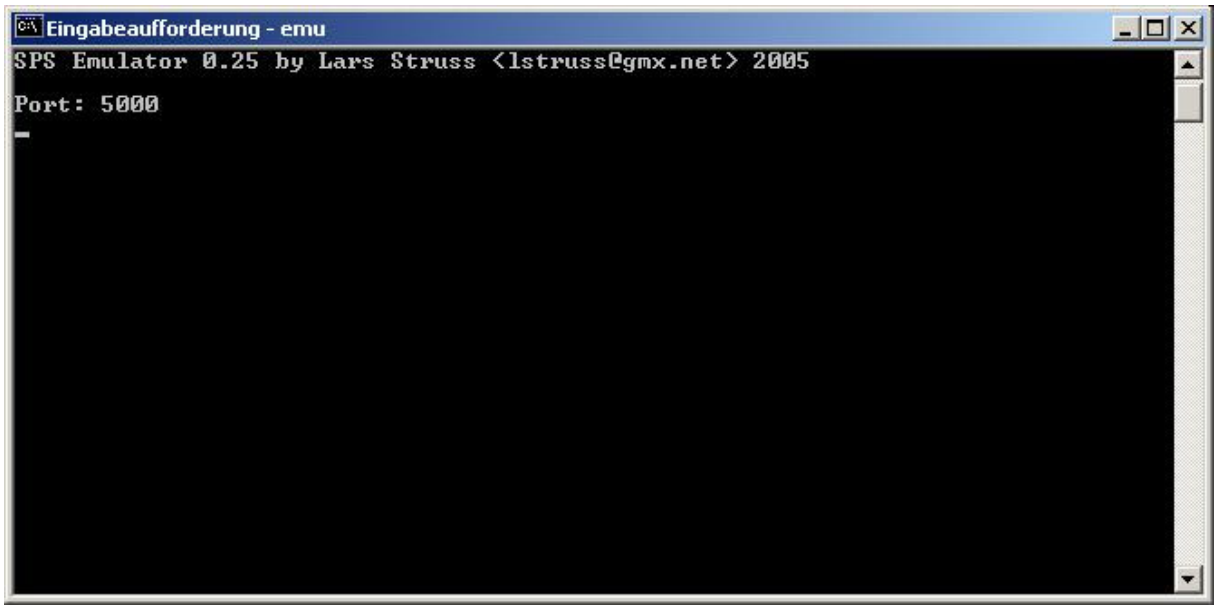


Bild 5.2.1-1: Textausgabe des Servers

Der Emulator wird gewöhnlich über die Konsole (siehe nächstes Kapitel) beendet. Er kann aber auch über die Tastenkombination STRG+C direkt beendet werden.

5.2.2. Starten der Konsole

Die Konsole wird analog durch Doppelklick auf das Skript ***cons.bat*** gestartet. Alternativ kann auch der Befehl ***java -jar console.jar*** im Emulatorverzeichnis eingegeben werden.

Es öffnet sich das Hauptfenster der Konsole:

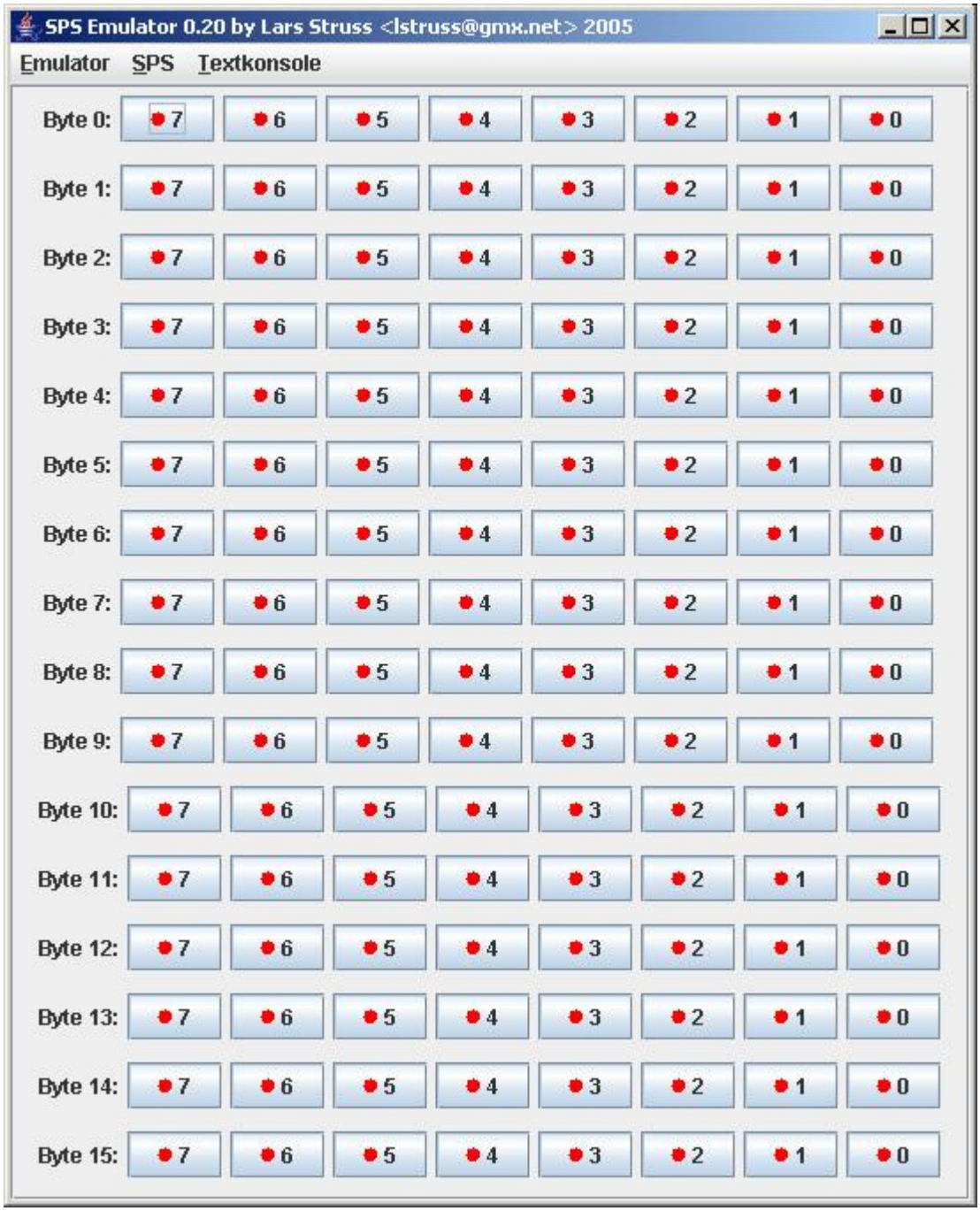


Bild 5.2.2-1: Hauptfenster der Konsole

Auf die Elemente des Konsolenfensters wird im Kapitel 5.3.6 detaillierter eingegangen.

5.2.3. Starten der Beispiel-Applikation „Fahrstuhl“

Im folgenden Abschnitt wird die Bedienung des Systems am Beispiel der Applikation „Fahrstuhl“ beschrieben.

Die Fahrstuhl-Applikation wird durch Doppelklick auf das Skript ***elev.bat*** gestartet. Alternativ kann auch der Befehl ***java -jar elevator.jar*** im Emulatorverzeichnis eingegeben werden.

Es erscheint ein Fenster mit einer symbolischen Darstellung eines Fahrstuhls:

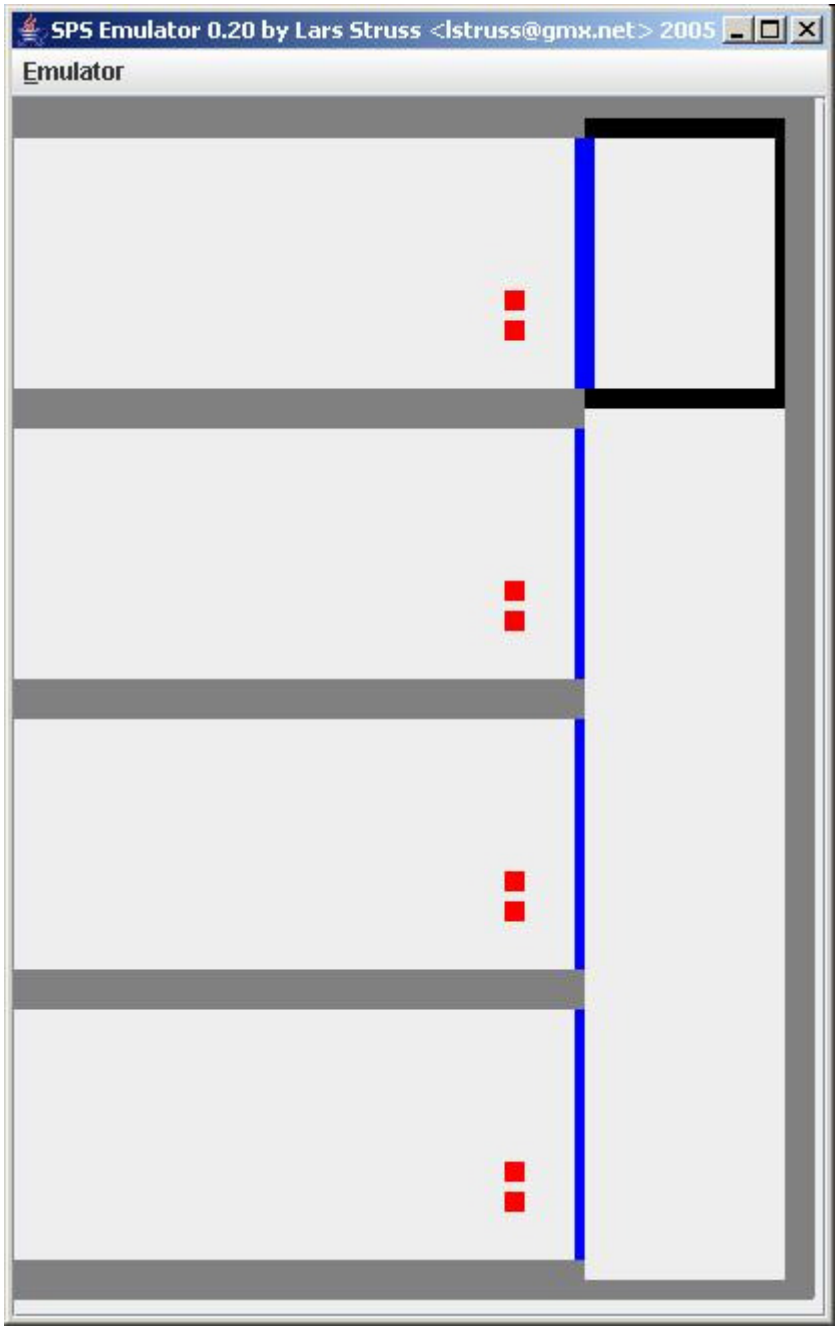


Bild 5.2.3-1: Fahrstuhl-Applikation

5.3. Erste Schritte

5.3.1. Zusammenschaltung aller Module zu einem virtuellem Schaltkreis

Nachdem alle drei Programme (Emulator, Konsole und Fahrstuhl-Applikation) gestartet sind, müssen diese nun zu einem „ganzen“ bzw. zu einem virtuellen „Schaltkreis“ zusammengeschlossen werden.

Der Emulator (emu.jar) arbeitet als Server, der seine Dienste den anderen Programmen über das Netzwerk zur Verfügung stellt. Er verfügt über keine eigene Benutzeroberfläche.

Sowohl die Konsole als auch die Applikationen verfügen im Menü „Emulator“ über einen Menüpunkt „Verbinden“. Dieser öffnet eine Anmelde-Dialogbox, in der die Netzwerkadresse eines laufenden Emulatorservers eingetragen werden muss. Die Felder „Anmeldename“ und „Passwort“ sind deaktiviert und können nicht ausgefüllt werden. Sie werden erst in einer zukünftigen Programmversion benutzt. In das Feld „Server“ muss die IP-Adresse oder der DNS-Name des Computers eingetragen werden, z.B. „x10.informatik.uni-bremen.de“. Die Vorgabe „127.0.0.1“ bezeichnet einen Emulatorserver, der auf dem gleichen Computer wie die Konsole/Applikation läuft. Der Port bezeichnet den TCP/IP-Port, auf dem der Server auf eingehende Verbindungen wartet. Dieser ist im Normalfall der Port 5000. Diese Einstellung kann jedoch mit dem Parameter „-p <portnummer>“ am Server verändert werden. In diesem Fall muss in das Feld „Port“ derselbe Wert wie beim „-p“-Parameter eingetragen werden.

Mit einem Klick auf die „Verbinden“-Schaltfläche baut die Konsole/Anwendung eine Verbindung zum Server auf. Wenn die Verbindung fehlschlägt, wird dies durch eine Dialogbox gemeldet. Der Fehlschlag kann die folgenden Ursachen haben:

- Die Parameter im Anmeldefenster sind ungültig oder enthalten einen Schreibfehler.
- Der Emulatorserver wurde nicht gestartet.
- Die Netzwerkverbindung wurde unterbrochen.
- Eine Firewall verhindert den Aufbau der Verbindung.

Nachdem Konsole und Fahrstuhl-Applikation auf diese Weise mit dem Server verbunden wurden, ist der (virtuelle) Schaltkreis vollständig. Nun kann die SPS programmiert werden.

5.3.2. Programmierung der SPS

Die Konsole verfügt im Menü „SPS“ über den Befehl „Programmbaustein laden“. Dieser ist ausgegraut, solange die Konsole nicht verbunden ist und wird aktiv, sobald eine Verbindung besteht. Wird der Menüpunkt angewählt, so öffnet sich ein Auswahlfenster,

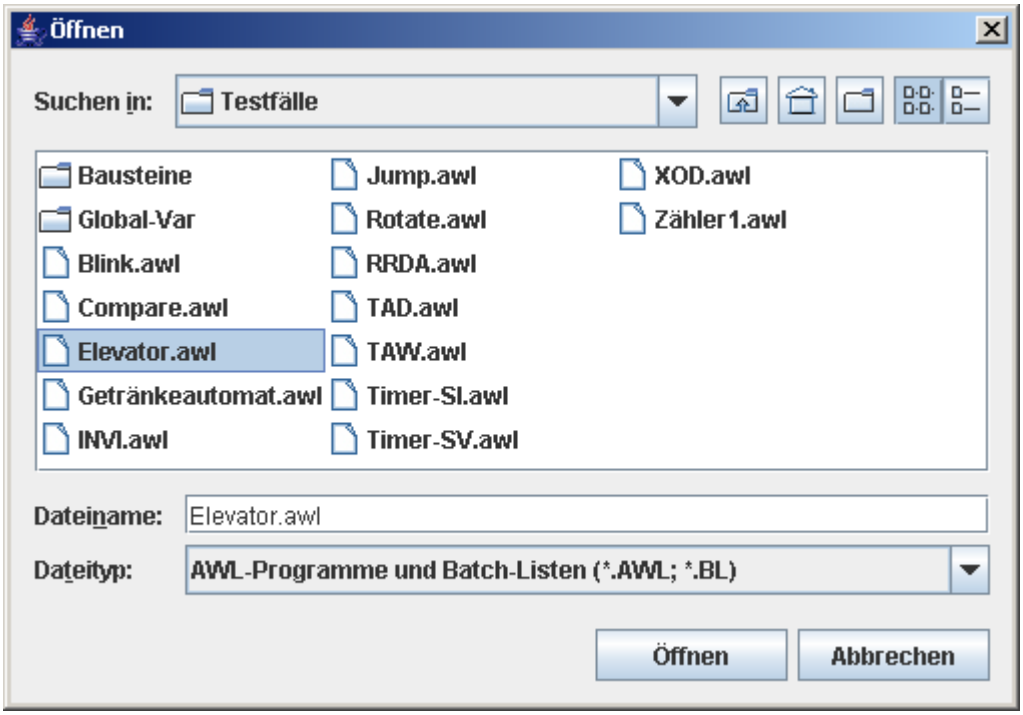


Bild 5.3.2-1: Dateiauswahldialogbox

das zwei Dateitypen zur Auswahl bietet: die AWL-Programme und die Batch-Listen. AWL-Programme sind Textdateien mit der Dateiendung .AWL. Sie enthalten den Programmcode und/oder Variablendeklarationen für genau einen SPS-Programmbaustein.
Häufig besteht ein SPS-Programm aus vielen Programmbausteinen, so dass es ziemlich umständlich ist, jeden Baustein einzeln in die SPS zu laden. Um das Laden größerer Projekte zu beschleunigen, gibt es daher die sogenannten Batch-Listen. Batch-Listen sind Textdateien mit der Endung .BL. Sie enthalten eine Liste (ein Dateiname pro Zeile) mit Programmbausteinen, die geladen werden sollen.

Wichtig: Sowohl beim manuellen als auch beim automatischen Laden von Programmbausteinen ist die Reihenfolge wichtig. Bevor ein Programmbaustein geladen werden kann, müssen zuvor alle Bausteine geladen werden, die von diesem Baustein aufgerufen werden. Andernfalls kann der AWL-Compiler nicht überprüfen, ob alle Bausteinparameter korrekt übergeben wurden. Er meldet gegebenenfalls, dass der betreffende Baustein unbekannt ist.

Zur Veranschaulichung des Programms im Rahmen dieser Tour sollte nun das mitgelieferte Beispielprogramm „Elevator.awl“ geladen werden. Es implementiert eine rudimentäre Fahrstuhlsteuerung.

5.3.3. Öffnen der Textkonsole

Über das Menü „Textkonsole“ → „Konsolenfenster öffnen“ kann ein Protokollfenster geöffnet werden, in dem alle eventuelle Fehlermeldungen vermerkt sind. Da Elevator.awl fehlerfrei sein sollte, sollte hier nach dem Laden „0 Fehler, 0 Warnungen.“ stehen.

5.3.4. Starten der SPS

Nun kann die SPS über den Menüpunkt „Kaltstart“ im Menü „SPS“ gestartet werden. Im Konsolenfenster sollten sofort zwei grüne LEDs aufleuchten, auf deren Bedeutung später noch eingegangen wird. Im Fenster der Fahrstuhlapplikation sollte der Fahrstuhl mit geöffneten Türen in der obersten Etage stehen.

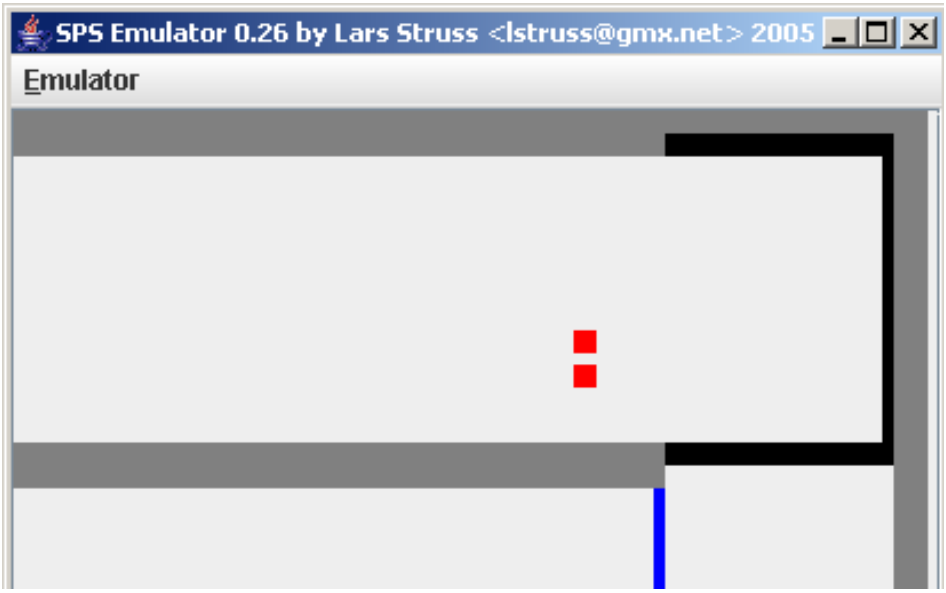


Bild 5.3.4-1: Fahrstuhl im obersten Geschoss, Türen geöffnet

5.3.5. Steuerung der Fahrstuhlkabine

Mit einem Mausklick auf einen der beiden roten Etagenschalter kann der Fahrstuhl in die betreffende Etage geholt werden.

5.3.6. Überwachung des Programms mit der Steuerkonsole

Parallel zur Fahrstuhlbewegung kann in der Konsole der Zustand der SPS-Ausgänge überwacht werden.

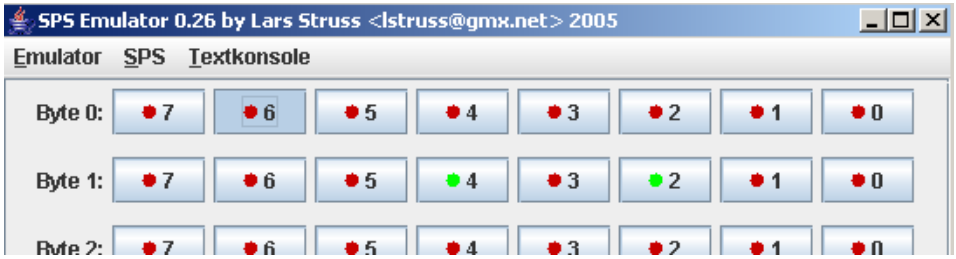


Bild 5.3.6-1: Eingang E0.6 sowie Ausgänge A1.2 und A1.4 sind gesetzt

Die Konsole enthält links eine Beschriftung der Ein-/Ausgangsbytes, beginnend bei Byte 0. Jedes Bit dieses Bytes besteht aus einer bläulichen Schaltfläche und einem farbigen Kreis als Zweifarb-LED im Inneren. Über die Schaltfläche können Eingänge gesetzt werden. Eine

gedrückte Schaltfläche steht dabei für einen logischen Highpegel an dem betreffenden Biteingang, eine nicht gedrückte für einen Lowpegel. Die „LED“ im inneren der Schaltflächen signalisiert die Ausgangsbits des jeweiligen Ausgangs. Eine grüne LED symbolisiert einen Highpegel, eine rote einen Lowpegel.

5.4. Übersicht der Menüfunktionen der Konsole

Hier eine Übersicht der Menüfunktionen der Konsole:

Menü:	Menüeintrag:	Erläuterung:
Emulator	Verbinden	Stellt eine Verbindung zu einem Emulatorserver her. Der Befehl ist ausgegraut, wenn bereits eine Verbindung besteht.
	Trennen	Trennt eine bestehende Serververbindung.
	Programm beenden und Server herunterfahren	Beendet die Konsole und fährt den Server herunter. Anschließend sind keine Zugriffe auf den Server mehr möglich, und das Serverprogramm muss zur erneuten Verwendung manuell gestartet werden. Wenn der Server über ein Skript gestartet wird, das den Server beim Beenden automatisch neu aufruft, kann dieser Befehl verwendet werden, um den Server ferngesteuert zu neu zu starten.
	Programm beenden	Beendet die Konsole.
SPS	Kaltstart	Startet die Programmausführung der SPS. Zur Zeit erfolgt noch keine Unterscheidung zwischen Warm- und Kaltstarts. Laufzeitfehler werden in der Textkonsole vermerkt.
	Wiederanlauf	
	Stop	Beendet die Programmausführung der SPS.
	Programmbaustein laden	Öffnet ein Dateiauswahlfenster zum Laden von Programmbausteinen. Auch wenn ein Laden von Bausteinen im laufenden Betrieb möglich ist, sollte die SPS zur Vermeidung von Laufzeitfehlern vor Änderungen am Programm angehalten werden. Alle vom zu ladenden Baustein referenzierten Bausteine müssen bereits geladen sein. Bei Änderungen an der Parameterliste einer Funktion sollten auch alle Bausteine, die diese Funktion aufrufen, neu geladen werden. Fehler beim Kompilieren des Bausteins werden in der Textkonsole vermerkt.
Textkonsole	Konsolenfenster öffnen	Öffnet die Textkonsole mit Status und Fehlermeldungen der letzten Operationen.
	Automatisch im Vordergrund	Öffnet die Textkonsole, sobald sich ihr Inhalt geändert hat.
	Signal bei neuen Meldungen	Erzeugt ein akustisches Signal, sobald sich der Inhalt der Textkonsole ändert. Diese Funktion ist in dieser Version noch nicht verfügbar.
	Löschen	Löscht den Text in der Textkonsole.

Die Befehle des Emulator-Menüs der Fahrstuhl-Applikation entsprechen denen im Menü der Konsole.

5.5. Unterstützter Befehlssatz

Der Emulator unterstützt die folgenden AWL-Befehle:

Befehl:	Erklärung:
ORGANIZATION_BLOCK	Deklariert einen Organisationsbaustein (OB)
FUNCTION_BLOCK	Deklariert einen Funktionsbaustein (FC oder FB)
DATA_BLOCK	Deklariert einen Datenbaustein (DB)
U	Logische UND-Verknüpfung
UN	Logische UND-NICHT-Verknüpfung
O	Logische ODER-Verknüpfung
ON	Logische ODER-NICHT-Verknüpfung
X	Logische ENTWEDER-ODER-Verknüpfung (XOR)
XN	Logische negierte ENTWEDER-ODER-Verknüpfung
U(	Logische UND-Verknüpfung mit Verzweigung
UN(	Logische UND-NICHT-Verknüpfung mit Verzweigung
O(	Logische ODER-Verknüpfung mit Verzweigung
ON(	Logische ODER-NICHT-Verknüpfung mit Verzweigung
X(	Logische ENTWEDER-ODER-Verknüpfung (XOR) mit Verzweigung
XN(	Logische negierte ENTWEDER-ODER-Verknüpfung mit Verzweigung
)	Ende der logischen Verzweigung
=	Kopiere VKE in Bit (Zuweisung)
S	Setze Bit, wenn VKE = 1
R	Lösche Bit, wenn VKE = 1
NOT	Invertiere VKE
SAVE	SAVE Sichere VKE im BIE-Bit
SET	Setze VKE
CLR	Lösche VKE
FP	Detektiere positive Flanke
FN	Detektiere negative Flanke
L	Lade Zähler/Zeitgeber/Byte/Wort/Doppelwort/Konstante in Akku
LC	Lade Zähler/Zeitgeber als BCD-Zahl in Akku
T	Transferiere Zähler/Zeitgeber/Byte/Wort/Doppelwort/Konstante in Akku
==I	Akku-Vergleich Ganzzahlen (16 Bit)
<>I	Akku-Vergleich Ganzzahlen (16 Bit)

>I	Akku-Vergleich Ganzzahlen (16 Bit)
<I	Akku-Vergleich Ganzzahlen (16 Bit)
>=I	Akku-Vergleich Ganzzahlen (16 Bit)
<=I	Akku-Vergleich Ganzzahlen (16 Bit)
==D	Akku-Vergleich Ganzzahlen (32 Bit)
<>D	Akku-Vergleich Ganzzahlen (32 Bit)
>D	Akku-Vergleich Ganzzahlen (32 Bit)
<D	Akku-Vergleich Ganzzahlen (32 Bit)
>=D	Akku-Vergleich Ganzzahlen (32 Bit)
<=D	Akku-Vergleich Ganzzahlen (32 Bit)
==R	Akku-Vergleich Gleitpunktzahlen (32 Bit)
<>R	Akku-Vergleich Gleitpunktzahlen (32 Bit)
>R	Akku-Vergleich Gleitpunktzahlen (32 Bit)
<R	Akku-Vergleich Gleitpunktzahlen (32 Bit)
>=R	Akku-Vergleich Gleitpunktzahlen (32 Bit)
<=R	Akku-Vergleich Gleitpunktzahlen (32 Bit)
BTI	BCD wandeln in Ganzzahl (16 Bit)
ITB	Ganzzahl (16 Bit) wandeln in BCD
BTD	BCD wandeln in Ganzzahl (32 Bit)
DTB	Ganzzahl (32 Bit) wandeln in BCD
ITD	Ganzzahl (16 Bit) wandeln in Ganzzahl (32 Bit)
DTR	Ganzzahl (32 Bit) wandeln in Gleitpunktzahl (32 Bit, IEEE-FP)
INVI	1-Komplement Ganzzahl (16 Bit)
INVD	1-Komplement Ganzzahl (32 Bit)
NEGI	2-Komplement Ganzzahl (16 Bit)
NEGD	2-Komplement Ganzzahl (32 Bit)
NEGR	Negiere Gleitpunktzahl (32 Bit, IEEE-FP)
TAW	Tausche Reihenfolge der Bytes im AKKU 1-L (16 Bit)
TAD	Tausche Reihenfolge der Bytes im AKKU 1 (32 Bit)
RND	Runden einer Gleitpunktzahl zur Ganzzahl
TRUNC	Runden einer Gleitpunktzahl durch Abschneiden
RND+	Runden einer Gleitpunktzahl zur nächsthöheren Ganzzahl
RND-	Runden einer Gleitpunktzahl zur nächstniederen Ganzzahl
TAK	Tausche AKKU 1 mit AKKU 2
PUSH	Schiebt alle Akku-Werte um einen Akku weiter (aufsteigend)
POP	Schiebt alle Akku-Werte um einen Akku weiter (absteigend)
ENT	Schiebt alle Akku-Werte außer AKKU 1 um einen Akku weiter (aufsteigend)
LEAVE	Schiebt alle Akku-Werte außer AKKU 1 um einen Akku weiter (absteigend)

ZV	Zählen vorwärts
ZR	Zählen rückwärts
SI	Zeit als Impuls
SV	Zeit als verlängerter Impuls
SE	Zeit als Einschaltverzögerung
SS	Zeit als speichernde Einschaltverzögerung
SA	Zeit als Ausschaltverzögerung
FR	Freigabe Zähler/Zeitgeber
UW	UND-Wort (16 Bit)
UD	UND-Doppelwort (32 Bit)
OW	ODER-Wort (16 Bit)
OD	ODER-Doppelwort (32 Bit)
XOW	EXKLUSIV-ODER-Wort (16 Bit)
XOD	EXKLUSIV-ODER-Doppelwort (32 Bit)
CALL	Bausteinaufruf (optional mit Parametern)
CC	Bedingter Bausteinaufruf (ohne Parameter)
UC	Bausteinaufruf (ohne Parameter)
BE	Bausteinende
BEB	Bausteinende bedingt
BEA	Bausteinende absolut
SPA	Springe absolut
SPL	Sprungleiste
SPB	Springe, wenn VKE = 1
SPBN	Springe, wenn VKE = 0
SPBB	Springe, wenn VKE = 1 und rette VKE ins BIE
SPBNB	Springe, wenn VKE = 0 und rette VKE ins BIE
SPBI	Springe, wenn BIE = 1
SPBIN	Springe, wenn BIE = 0
SPO	Springe, wenn OV = 1
SPS	Springe, wenn OS = 1
SPZ	Springe, wenn Ergebnis = 0
SPN	Springe, wenn Ergebnis <> 0
SPP	Springe, wenn Ergebnis > 0
SPM	Springe, wenn Ergebnis < 0
SPPZ	Springe, wenn Ergebnis >= 0
SPMZ	Springe, wenn Ergebnis <= 0
SPU	Springe, wenn Ergebnis ungültig
LOOP	Programmschleife mit Schleifenzähler
+I	Addiere AKKU 1 und 2 als Ganzzahl (16 Bit)
-I	Subtrahiere AKKU 1 von 2 als Ganzzahl (16 Bit)

*I	Multipliziere AKKU 1 und 2 als Ganzzahl (16 Bit)
/I	Dividiere AKKU 2 durch 1 als Ganzzahl mit Rest (16 Bit)
+D	Addiere AKKU 1 und 2 als Ganzzahl (32 Bit)
-D	Subtrahiere AKKU 1 von 2 als Ganzzahl (32 Bit)
*D	Multipliziere AKKU 1 und 2 als Ganzzahl (32 Bit)
/D	Dividiere AKKU 2 durch 1 als Ganzzahl (32 Bit)
MOD	Divisionsrest Ganzzahl (32 Bit)
+R	Addiere AKKU 1 und 2 als Gleitpunktzahl (32 Bit)
-R	Subtrahiere AKKU 1 von 2 als Gleitpunktzahl (32 Bit)
*R	Multipliziere AKKU 1 und 2 als Gleitpunktzahl (32 Bit)
/R	Dividiere AKKU 2 durch 1 als Gleitpunktzahl (32 Bit)
ABS	Absolutwert einer Gleitpunktzahl (32 Bit, IEEE-FP)
SQR	Bilden des Quadrats einer Gleitpunktzahl (32 Bit)
SQRT	Bilden der Quadratwurzel einer Gleitpunktzahl (32 Bit)
EXP	Bilden des Exponentialwerts einer Gleitpunktzahl (32 Bit)
LN	Bilden des natürlichen Logarithmus einer Gleitpunktzahl (32 Bit)
SIN	Bilden des Sinus eines Winkels als Gleitpunktzahlen (32 Bit)
COS	Bilden des Cosinus eines Winkels als Gleitpunktzahlen (32 Bit)
TAN	Bilden des Tangens eines Winkels als Gleitpunktzahlen (32 Bit)
ASIN	Bilden des Arcussinus einer Gleitpunktzahl (32 Bit)
ACOS	Bilden des Arcuscosinus einer Gleitpunktzahl (32 Bit)
ATAN	Bilden des Arcustangens einer Gleitpunktzahl (32 Bit)
RLDA	Rotiere Akku 1 links über A1-Bit (32 Bit)
RRDA	Rotiere Akku 1 rechts über A1-Bit (32 Bit)
SSI	Schiebe Vorzeichen rechts Ganzzahl (16 Bit)
SSD	Schiebe Vorzeichen rechts Ganzzahl (32 Bit)
SLW	Schiebe links Wort (16 Bit)
SRW	Schiebe rechts Wort (16 Bit)
SLD	Schiebe links Doppelwort (32 Bit)
SRD	Schiebe rechts Doppelwort (32 Bit)
RLD	Rotiere links Doppelwort (32 Bit)
RRD	Rotiere rechts Doppelwort (32 Bit)
VAR	Beginnt einen Deklarationsblock für lokale Variablen
VAR_GLOBAL	Beginnt einen Deklarationsblock für globale Variablen
VAR_INPUT	Beginnt einen Deklarationsblock für Eingangsparameter
VAR_OUTPUT	Beginnt einen Deklarationsblock für Ausgangsparameter
END_VAR	Beendet einen Variablen-Deklarationsblock

Eine detaillierte Beschreibung aller Befehle befindet sich in [Siemens01].

5.6. Unterstützte Formate für Konstanten

Der Emulator unterstützt die folgenden Notationen für numerische Konstanten:

Format:	Beispielcode:
Dezimalzahl (16 Bit)	L –10
Dezimalzahl (32 Bit)	L#50000
hexadezimalen Zahl (8 Bit)	L B#16#FF
hexadezimalen Zahl (16 Bit)	L W#16#FFFF
hexadezimalen Zahl (32 Bit)	L DW#16#FFFFFFFF
Dual-Zahl (16 Bit)	L 2#11111111
Dual-Zahl (32 Bit)	L 2#1111111111111111
Gleitpunktzahl	L 1.12345E10
Zählerkonstante	L C#999
Zeitkonstante	L S5T#2M10S

5.7. Beispielprogramme und Testfälle

Im Verzeichnis „Testfälle“ befinden sich verschiedene Beispielprogramme, die die Benutzung des Simulators veranschaulichen und zur Überprüfung seiner Funktionsweise benutzt werden können.

5.7.1. Bausteine

Funktion: Testet Bausteinaufrufe über CALL, CC, UC.
Dateien: FB1.awl, DB1.awl, DB2.awl, FC1.awl, FC2.awl, FC3.awl, OB1.awl
Benutzte Eingänge: E4.0, E4.1, E4.2, E4.3, E4.7, EW0, EW2
Benutzte Ausgänge: A4.0, A4.2, A4.3, A4.7, AW0, AW2
Beschreibung:

- A4.0 ← E4.0 XOR E4.1
- A4.2 ← E4.2
- A4.3 ← E4.3
- A4.7 blinkt wenn E4.7 gesetzt ist.
- AW0 ← EW0 XOR 0x000F
- AW2 ← EW2 XOR 0x00F0

5.7.2. Bausteine2

Funktion: Testet die Speicherverwaltung bei Bausteinaufrufen mit falscher Parameterzuordnung.
Dateien: FC1.awl, OB1.awl
Benutzte Eingänge: -
Benutzte Ausgänge: -
Beschreibung:
Dieses Programm bricht mit einer Fehlermeldung ab.

5.7.3. Global-Var

Funktion: Demonstriert die Verwendung von globalen Variablen.
Dateien: FC1.awl, FC2.awl, OB1.awl
Benutzte Eingänge: E0.0, E0.1
Benutzte Ausgänge: A0.0, A0.1
Beschreibung:

- A0.0 ← E0.0 (via globale Variable und zwei Bausteinaufrufe)
- A0.1 ← E0.1 (direkt)

5.7.4. Blink

Funktion: einfaches Blinklicht
Dateien: Blink.awl
Benutzte Eingänge: -
Benutzte Ausgänge: A0.0, AW4
Beschreibung:

- A0.0 blinkt
- Auf AW4 wird zur Kontrolle der aktuelle Wert des Zeitgebers ausgegeben

5.7.5. Clear

Funktion: Setzt alle Ausgänge zurück.
Dateien: Clear.awl
Benutzte Eingänge: -
Benutzte Ausgänge: AD0, AD4, AD8, AD12
Beschreibung:
Dieses Programm setzt alle SPS-Ausgänge zurück. Es ist nützlich, um die SPS nach einem Programmwechsel neu zu initialisieren.

5.7.6. Compare

Funktion: Vergleicht zwei Zahlen.
Dateien: Compare.awl
Benutzte Eingänge: ED0, ED4
Benutzte Ausgänge: A0.0, A0.1, A0.2, A0.5, A0.6, A0.7, A1.0, A1.1, A1.2
Beschreibung:
Das Programm liest die Zahlen an den Doppelwort-Eingängen ED0 und ED4. Abhängig vom Vergleich werden die Ausgänge gesetzt:

Ausgang:	Gesetzt wenn (andernfalls zurückgesetzt):
A0.0	EW0 < EW4
A0.1	EW0 = EW4
A0.2	EW0 > EW4
A0.5	EW0 < EW4 (via Flags)
A0.6	EW0 = EW4 (via Flags)
A0.7	EW0 > EW4 (via Flags)
A1.0	ED0 < ED4
A1.1	ED0 = ED4

A1.2	ED0 > ED4

5.7.7. Elevator

Funktion: Einfaches Beispielprogramm für die Fahrstuhl-Applikation
Dateien: Elevator.awl
Benutzte Eingänge: E0.0, E0.1 E0.2, E0.3, E2.0, E2.1, E2.2, E2.3, E2.4, E2.5, E2.6, E2.7
Benutzte Ausgänge: A1.0, A1.1, A1.2, A1.4, A1.5, A1.6, A1.7, A3.0, A3.1, A3.2, A3.3, A3.4, A3.5, A3.6, A3.7
Beschreibung:
Das Programm wird durch die roten Fahrstuhl-Etagentasten in der Fahrstuhlapplikation gesteuert. Die Kabine hält in der gewünschten Etage, und die Kabinen sowie Etagentüren werden geöffnet und geschlossen.

5.7.8. Getränkeautomat

Funktion: Steuerprogramm für einen einfachen Getränkeautomaten
Dateien: Getränkeautomat.awl
Benutzte Eingänge: siehe [Struß01]
Benutzte Ausgänge: siehe [Struß01]
Beschreibung:
Dieses Programm steuert den von mir im Sommersemester 2001 in Produktionsinformatik entwickelten Lego-Getränkeautomaten.

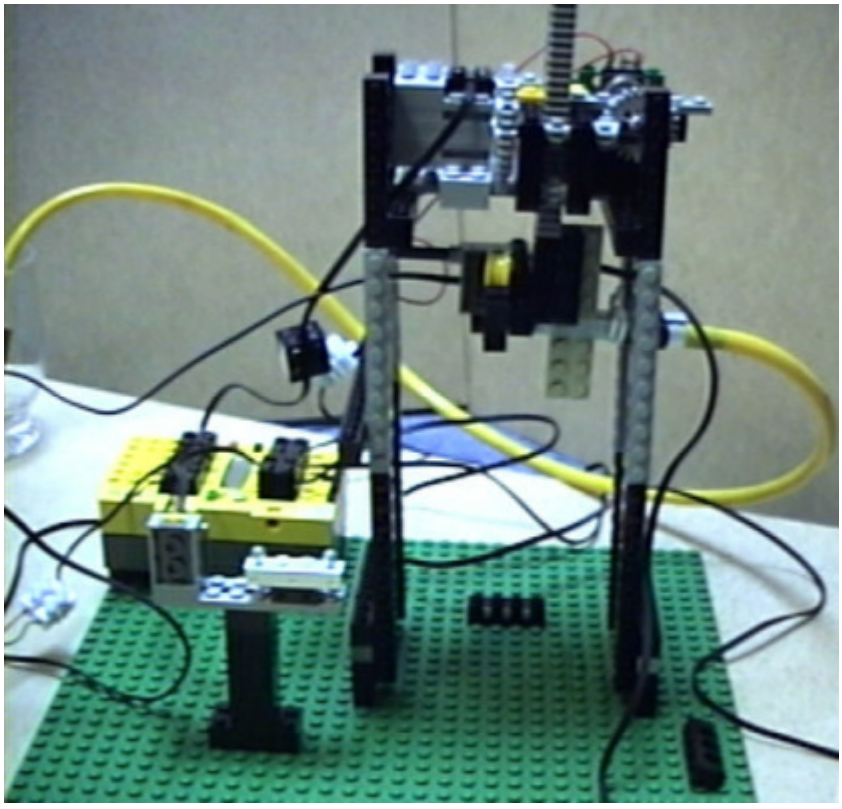


Bild 5.7-1: Lego-Getränkeautomat

Das Projekt ist unter siehe [Struß01] dokumentiert.

5.7.9. INVI

Funktion: Invertiert eine Ganzzahl.
Dateien: INVI.awl
Benutzte Eingänge: EW0
Benutzte Ausgänge: AW0, AW2
Beschreibung:
Das Programm liest den Eingang EW0 und gibt den Wert zur Kontrolle auf AW0 aus. Anschließend wird er invertiert und auf dem Ausgang AW2 ausgegeben.

5.7.10. Jump

Funktion: Testprogramm für Sprungbefehle
Dateien: Jump.awl
Benutzte Eingänge: EB0, E1.0
Benutzte Ausgänge: AB0, A1.0, A1.1, A1.2, A1.3, A1.7
Beschreibung:
Das Programm liest EB0 und kopiert den Wert, falls E1.0 gesetzt ist, in AB0. Anschließend werden mit einer Sprungliste (SPL) die unteren Bit zwei Bits ausgewertet:

EB0	A1.0	A1.1	A1.2	A1.3	A1.7
0	1	0	0	0	0
1	0	1	0	0	0
2	0	0	1	0	0
>2	0	0	0	0	1

5.7.11. Konstanten

Funktion: Testprogramm für Konstantenformate
Dateien: Konstanten.awl
Benutzte Eingänge: -
Benutzte Ausgänge: -
Beschreibung:
Lädt verschiedene Konstanten in den Akku, keine zur Laufzeit sichtbare Funktion.

5.7.12. Rotate

Funktion: Testprogramm für Rotationsbefehle
Dateien: Rotate.awl
Benutzte Eingänge: ED0, EB4
Benutzte Ausgänge: AD0, A4.0
Beschreibung:
Rotiert den Inhalt von ED0 um EB4 Bits weiter. Das Ergebnis wird über AD0 ausgegeben. Das A1-Bit (enthält das zuletzt rotierte Bit) wird nach A4.0 kopiert.

5.7.13. RRDA

Funktion: Erweitertes Testprogramm für Rotationsbefehle mit Schleife und Variablen
Dateien: RRDA.awl
Benutzte Eingänge: ED0, EB4

Benutzte Ausgänge: AD0, A4.0
Beschreibung:
Verschiebt Doppelwort aus ED0 um EB4 Bitstellen. Die Rotation erfolgt über das A1-Bit., welche an Ausgang A4.0 anliegt. EB4=0 entspricht 256 Schleifendurchläufen. Das Programm reagiert dadurch etwas träge.

5.7.14. TAD

Funktion: Testprogramm für TAD-Instruktion
Dateien: TAD.awl
Benutzte Eingänge: ED0
Benutzte Ausgänge: AD0, AD4
Beschreibung:
Das Programm liest das Doppelwort ED0 in den Akku und gibt es auf AD0 aus. Anschließend werden die Bytes im Akku (in ihrer Reihenfolge) umgedreht und auf AD4 ausgegeben.

5.7.15. TAW

Funktion: Testprogramm für TAW-Instruktion
Dateien: TAW.awl
Benutzte Eingänge: EW0
Benutzte Ausgänge: AW0, AW2
Beschreibung:
Das Programm liest das Wort EW0 in den Akku und gibt es auf AW0 aus. Anschließend werden die Bytes im Akku vertauscht und auf AW2 ausgegeben.

5.7.16. Timer-SI

Funktion: Timer mit Zeit als Impuls
Dateien: Timer-SI.awl
Benutzte Eingänge: E0.0, E0.1
Benutzte Ausgänge: A0.0, A0.1, AW1
Beschreibung:
Über Eingang E0.0 kann ein Zeitgeber gestartet werden, E0.1 setzt den Zeitgeber wieder zurück. Solange der Zeitgeber läuft, ist der Ausgang A0.0 gesetzt, andernfalls A0.1. Über AW1 wird der aktuelle Zeitgeberwert zur Kontrolle ausgegeben.

5.7.17. Timer-SV

Funktion: Timer mit Zeit als verlängertem Impuls
Dateien: Timer-SV.awl
Benutzte Eingänge: E0.0, E0.1
Benutzte Ausgänge: A0.0, A0.1, AW1
Beschreibung:
Über Eingang E0.0 kann ein Zeitgeber gestartet werden, E0.1 setzt den Zeitgeber wieder zurück. Solange der Zeitgeber läuft, ist der Ausgang A0.0 gesetzt, andernfalls A0.1. Über AW1 wird der aktuelle Zeitgeberwert zur Kontrolle ausgegeben.

5.7.18. XOD

Funktion: Doppelwort-XOR
Dateien: XOD.awl
Benutzte Eingänge: ED0, ED4
Benutzte Ausgänge: AD12
Beschreibung:
AD12 ← ED0 XOR ED4

5.7.19. Zähler1

Funktion: Zähler-Test 1
Dateien: Zähler1.awl
Benutzte Eingänge: E0.0, E0.1, E0.2, E0.7
Benutzte Ausgänge: AB1
Beschreibung:

Eingang:	Funktion:
E0.0	Zählt vorwärts
E0.1	Zählt rückwärts
E0.2	Setzt Zähler auf 12
E0.7	Setzt Zähler auf 0 (Reset)

Der Zählerstand wird über AB0 ausgegeben.

5.7.20. Zähler2

Funktion: Zähler-Test 2
Dateien: Zähler2.awl
Benutzte Eingänge: E0.0, E0.1, E0.2, E0.7
Benutzte Ausgänge: AB1
Beschreibung:

Eingang:	Funktion:
E0.0	Zählt vorwärts
E0.1	Zählt ebenfalls vorwärts
E0.2	Setzt Zähler auf 12.
E0.7	Setzt Zähler auf 0 (Reset)

Der Zählerstand wird über AB0 ausgegeben.

Der Zähler läuft durch, sobald einer der beiden Zähleingänge gesetzt wird und bleibt stehen, wenn beide Eingänge gesetzt werden. Dies entspricht dem Verhalten des Programms im Emulator WinSPS-S7.

6. Fazit und Ausblick

Vergleicht man das entstandene Programmpaket mit der Aufgabenstellung und der Anforderungsdefinition, so wurden alle gesteckten Ziele erreicht und teilweise sogar übertroffen. Gerade die offene Architektur der Software bietet viele Möglichkeiten, die von den genannten Anwendungsszenarien nur zum Teil ausgeschöpft werden.

Kompliziertester Teil der Implementierung war die Bausteinverwaltung und die Realisierung der Aufrufe zwischen Bausteinen mit Parameterübergabe. Während sich die meisten Programmteile sehr gut kapseln lassen, sind speziell bei der Parameterübergabe Interaktionen zwischen einer Vielzahl der Programmkomponenten nötig, was sehr leicht zu Synchronisationsproblemen und unerwünschten Seiteneffekten führen kann.

Die Lernplattform bietet eine solide Grundlage für weitere Verbesserungen und Entwicklungen. Die im Kapitel 4.5. aufgezeigten Grenzen des Programms bieten Anregungen für neue Entwicklungen: Eine Benutzerverwaltung ist bei der Anwendung im öffentlich zugänglichen Internet sicherlich bei umfangreicher Nutzung unumgänglich. Der Emulator-Kern könnte dann auch eine grafische Oberfläche erhalten um das komfortable Verwalten von Benutzerkonten und aktiven Verbindungen zu vereinfachen. Dennoch sollte der jetzige Textmodus beibehalten werden, da der Server so sehr einfach über Programme wie ssh fernadministriert werden kann.

Außerdem könnte die Steuerkonsole mit einem Quellcode-Editor und einem Code-Debugger aufgewertet werden, der Fehler direkt im Quelltext anzeigt. Außerdem könnte der Befehlssatz vervollständigt und der Emulator-Kern noch weiter modularisiert werden, so dass neben AWL-Programmen auch grafische Programme in Funktionsplan- oder Kontaktplan-Notation ausgeführt werden können bzw. diese automatisch in AWL-Programme umgewandelt werden können.

7. Literaturverzeichnis

[Bruns01] Bruns, F. W. et al (2002): Distributed Real and Virtual Learning Environment for Mechatronics and Tele-Service.
Final Report of EU-Project DERIVE, artec-paper 102, Bremen
http://arteclab.artec.uni-bremen.de/publications/artec-02-Bruns-artec-paper102DERIVEFinalReport.pdf vom 2005-09-15

[Bruns02] Bruns, F. W. (2003): Lernen in Mixed Reality.
ABWF (Ed.): Kompetenzentwicklung 2003, Waxmann, Berlin, S. 71-112
http://arteclab.artec.uni-bremen.de/publications/artec-03-Bruns-LernenInMixedReality.pdf vom 2005-09-15

[Eilers01] Eilers, H.: Eilers.net - Spass am Internet - SPS4Linux
http://www.eilers.net/sps/ vom 2005-09-19

[Festo01] Festo: Festo Didactic/Produkte/Software & E-Learning/EasyVeep
http://www.festo.com/didactic/shop.asp?action=detail&back=search&art_id=3915&view=1&e0=486&e1=853&actpage=1&sLevel=-1&sid=e894bbbce83591437d0bc23a1d327a7f&nation=de&lang=de vom 2005-09-19

[Lego01] LEGO Group, The: LEGO.com Mindstorms Home
http://mindstorms.lego.com/ vom 2005-09-19

[MatPLC01] MatPLC team: MatPLC home page
http://mat.sourceforge.net/ vom 2005-09-19

[MHJ01] MHJ-Software: MHJ-Software
http://www.mhj-software.com/de/ vom 2005-09-19

[Michaelides01] Michaelides, I.; Eleftheriou, P.; Müller, D (2004): A remotely accessible solar energy laboratory - A distributed learning experience.
1st Remote Engineering and Virtual Instrumentation International Symposium (REV'04), Villach (Austria), 28 - 29 September 2004
http://www.marvel.uni-bremen.de/fileadmin/templates/MARVEL/documents/2004-09-REV04-HTI-paper.pdf vom 2005-09-15

[Mono01] Mono-Project: Main Page - Mono
http://www.mono-project.com/ vom 2005-09-19

[Müller01] Müller, D., Bruns, W. (2004): Arbeitsprozessorientiertes Lernen in Mixed Reality Umgebungen.
In: Pangalos, J.; Spöttl, G.; Knutzen, S.; Howe, F. (Hrsg): Informatisierung von Arbeit, Technik und Bildung: Kurzfassung der Konferenzbeiträge; GTW-Herbstkonferenz, 04./05. Oktober 2004. triass druck & media KG, Hamburg 2004, S. 184-188.
http://www.marvel.uni-bremen.de/fileadmin/templates/MARVEL/documents/artec-04-MuellerBruns-Apo-Lernen-in-MR.pdf vom 2005-09-15

[Müller02] Müller, D., Bruns, F. W. (2005): Arbeitsprozessorientiertes Lernen in Mixed Reality Umgebungen.
In: Pangalos, J.; Spöttl, G.; Knutzen, S.; Howe, F. (Hrsg): Informatisierung von Arbeit, Technik und Bildung. (in print)
http://www.marvel.uni-bremen.de/fileadmin/templates/MARVEL/documents/2005-02-mueller-bruns-GTW.pdf vom 2005-09-19

[Müller03] Müller, D.; Bruns, F. W. (2005): Experiential Learning of Mechatronics in a Mixed Reality Learning Space.
Paper to be presented at the EDEN 2005 Conference, Finland. June 20-23, 2005
http://www.marvel.uni-bremen.de/fileadmin/templates/MARVEL/documents/2005-06-EDEN05-artec.pdf vom 2005-09-15

[Müller04] Müller, D. (2005): Zwischen Realem und Virtuellem - Mixed-Reality in der technischen Bildung.
In K. Lehmann & M. Schetsche (Hrsg.), Die Google-Gesellschaft. transcript-Verlag: Bielefeld.
http://www.marvel.uni-bremen.de/fileadmin/templates/MARVEL/documents/2005-04-mueller-google.pdf vom 2005-09-15

[Schäfer01] Schäfer, K., Bruns, F. W. (2001): PLC-Programming by Demonstration with graspable Models.
Proceedings of 6th IFAC Symposium on Cost Oriented Automation. Berlin, 8.-9. 10. 2001
http://arteclab.artec.uni-bremen.de/publications/artec-01-SchaeferBruns-PLCProgrammingbyDemo.pdf vom 2005-09-15

[Severing01] Severing, E.(2003): Anforderungen an eine Didaktik des E-Learning in der betrieblichen
Bildung. In: Dehnbostel, P. et al.: Perspektiven moderner Berufsbildung.
Bertelsmann, Bielefeld
http://www.f-bb.de/f-bbv9/downloads/Anforderungen_an_eine_Didaktik_des_E-Learning_in_der_betrieb.pdf vom 2005-09-18

[Siemens01] Siemens AG: Anweisungsliste (AWL) für S7-300&400 Referenzhandbuch
Ausgabe 01/2004, Bestellnummer 6ES7810-4CA07-8AW1
http://support.automation.siemens.com/WW/llisapi.dll/18653496?func=ll&objId=18653496&objAction=csView&lang=de&siteid=cseus&aktprim=0 vom 2005-09-19

[Siemens02] Siemens AG: Operationsliste S7-300 CPU 312 IFM, 314 IFM, 313, 314, 315, 315-2 DP, 316-2 DP, 318-2
Ausgabe 2, Bestellnummer 6ES7 398-8AA03-8AN0

[Siemens03] Siemens AG: SIMATIC Erste Schritte und Übungen mit STEP 7 V5.3 Getting Started
Ausgabe 01/2004, Bestellnummer 6ES7810-4CA07-8AW0
http://support.automation.siemens.com/WW/llisapi.dll/18652511?func=ll&objId=18652511&objAction=csView&lang=de&siteid=cseus&aktprim=0 vom 2005-09-19

[Siemens04] Siemens AG: SIMATIC Programmieren mit STEP 7 V5.3 Handbuch
Ausgabe 01/2004, Bestellnummer 6ES7810-4CA07-8AW0

http://support.automation.siemens.com/WW/llisapi.dll/18652056?func=ll&objId=18652056&objAction=csView&lang=de&siteid=cseus&aktprim=0 vom 2005-09-19

[Siemens05] Siemens AG: SIMATIC S7-200 Automatisierungssystem Systemhandbuch
Ausgabe 06/2004, Bestellnummer 6ES7298-8FA24-8AH0

[Siemens06] Siemens AG: SIMATIC Statement List (STL) for S7-300 and S7-400 Programming Reference Manual
Ausgabe 01/2004, Bestellnummer 6ES7810-4CA07-8BW1
http://support.automation.siemens.com/WW/llisapi.dll/18653496?func=ll&objId=18653496&objAction=csView&lang=en&siteid=cseus&aktprim=0 vom 2005-09-19

[Siemens07] Siemens AG: SIMATIC STEP 7 Von S5 nach S7 Umsteigerhandbuch
Ausgabe 01/2004, Bestellnummer 6ES7810-4CA07-8AW0
http://support.automation.siemens.com/WW/llisapi.dll/1118413?func=ll&objId=1118413&objAction=csView&lang=de&siteid=cseus&aktprim=0 vom 2005-09-19

[Siemens08] Siemens AG: SIMATIC Systemsoftware für S7-300/400 System- und Standardfunktionen Referenzhandbuch
Ausgabe 01/2004, Bestellnummer 6ES7810-4CA07-8AW1

[Siemens09] Siemens AG: SIMATIC Systemsoftware für S7-300/400 System- und Standardfunktionen Referenzhandbuch
Ausgabe 07/2005, Bestellnummer 6ES7810-4CA07-8AW1
http://support.automation.siemens.com/WW/llisapi.dll/1214574?func=ll&objId=1214574&objAction=csView&lang=de&siteid=cseus&aktprim=0 vom 2005-09-19

[Siemens10] Siemens AG: SIMATIC Systemsoftware für S7-300/400 Standardfunktionen Teil 2 Referenzhandbuch
Ausgabe 03/2000
http://support.automation.siemens.com/WW/llisapi.dll/1214904?func=ll&objId=1214904&objAction=csView&lang=de&siteid=cseus&aktprim=0 vom 2005-09-19

[Struß01] Struß, L.: Getränkeautomat mit Lego Mindstorms und WinSPS
http://www.tzi.de/~lst/ls/automat/index.html vom 2005-09-19
Die beiliegende CD enthält eine Kopie dieser Seite.

[Sun01] Sun Microsystems, Inc.: Java Communications API
http://java.sun.com/products/javacomm/index.jsp vom 2005-09-19

[Sun02] Sun Microsystems, Inc.: Java Native Interface
http://java.sun.com/docs/books/tutorial/native1.1/ vom 2005-09-19

[Sun03] Sun Microsystems, Inc.: Java Remote Method Invocation (Java RMI)
http://java.sun.com/products/jdk/rmi/ vom 2005-09-19

[Wiki01] Wikipedia: Speicherprogrammierbare Steuerung
http://de.wikipedia.org/wiki/Speicherprogrammierbare_Steuerung vom 2005-09-19

[Wellenreuther01] Wellenreuther, G., Zastrow, D. (2002): Automatisieren mit SPS - Theorie und Praxis.
Vieweg, Wiesbaden, 2. Auflage

8. Anhang

8.1. Inhaltsverzeichnis der CD-ROM

Die CD-ROM enthält die folgenden Verzeichnisse:

Verzeichnis	Zielgruppe:	Inhalt:
Getränkeautomat	Benutzer	Beschreibung, Bilder und Videos des in Kapitel 5.7.8. referenzierten Lego-Getränkeautomaten. Der Getränkeautomat wurde von mir im Rahmen der Produktionsinformatik-Veranstaltung im Sommersemester 2001 in Produktionsinformatik entwickelten und dient in dieser Diplomarbeit lediglich als Beispiel.
Java-Quellcode	Entwickler	Der vollständige Source-Code der SPS-Lernplattform für eigene Modifikationen.
Java-Quellcode-Dokumentation	Entwickler	Die JavaDoc-Dokumentation zum Source-Code der SPS-Lernplattform.
Java-vorkompiliert	Benutzer	vorkompilierte JAR-Dateien zum direkten Ausführen.
Testfälle	Benutzer	Die im Kapitel 5.7. beschriebenen Testfälle.
Text	Benutzer	Dieses Dokument in digitaler Form.

8.2. Lizenz

Die Software dieser Diplomarbeit ist unter der GNU GENERAL PUBLIC LICENSE (GPL) lizenziert. Eine Kopie dieser Lizenz befindet sich zusammen mit der Software auf der beiliegenden CD-ROM.

8.3. Glossar

- **Anweisungsliste:** Die Anweisungsliste (AWL) ist eine sehr bekannte Programmiersprache für SPS-Anwendungen. Sie wurde von der Firma Siemens für ihre SPS-Produkte entwickelt, wurde jedoch inzwischen auch von anderen Herstellern übernommen und ist damit ein Defacto-Standard in der Automatisierungstechnik.

- **Applikation:** Eine Anwendung (Client), die die Dienste des Emulator-Kerns (Server) in Anspruch nimmt. Applikationen ermöglichen die Steuerung der simulierten SPS (siehe Konsole), die Kommunikation mit realer Hardware über Hardware-Interfaces (siehe Treiber) oder der Bereitstellung virtueller Hardware, die über ein SPS-Programm programmiert werden kann.

- **Ausgangsbyte:** Eine Gruppe von 8 aufeinanderfolgenden Ausgängen. So bilden die Ausgänge A0.0 bis A0.7 das Ausgangsbyte AB0, die Ausgänge A1.0 bis A1.7 das Ausgangsbyte AB1 und so weiter.

- **Ausgangswort:** Zwei aufeinanderfolgende Ausgangsbytes. Das Ausgangswort AW0 besteht z.B. aus den beiden Ausgangsbytes AB0 und AB1.

- **AWL:** siehe Anweisungsliste

- **Baustein:** Ein Baustein ist ein modulares Stück AWL-Code, welches von anderen Bausteinen aufgerufen werden kann. Die Siemens Simatic-SPS unterscheiden verschiedene Art von Bausteinen, z.B. die Organisatonsbausteine (OB), (System-) Funktionsbausteine (FC/FB/SFC/SFB) und Datenbausteine. Siehe auch SFC.

- **BCD-Zahl:** Eine Zahl, die als binär codierte Dezimalzahl kodiert ist. Die BCD-Notation teilt jeder Ziffer einer Dezimalzahl eine Gruppe von 4 Bits zu, die diese Ziffer repräsentieren. Da diese Notation nicht alle möglichen Bitkombinationen verwendet, ist der darstellbare Zahlenbereich im BCD-Format verglichen mit der Dualzahldarstellung bei gleicher Bitanzahl geringer. BCD-Zahlen lassen sich jedoch aufgrund der nicht nötigen Umrechnung der Zahlenbasis mit sehr wenig Hardwareaufwand als Dezimalzahlen ausgeben, weshalb die BCD-Notatation in einigen Anwendungsbereichen sehr verbreitet ist. Weitere Informationen unter http://de.wikipedia.org/wiki/BCD .

- **Bibliothek:** Eine (Software-)Bibliothek enthält Programmteile und Daten, die für verschiedene Programme relevant sind. Bibliotheken sparen Speicherplatz, weil sich mehrere Programme den gleichen Programmcode teilen können. Sie vereinfachen die Programmierung, da das Rad nicht für jedes Programm neu erfunden werden muss. Einige speicherprogrammierbare Steuerungen bieten in Form von Systemfunktionen (SFCs) fertige Programmroutinen an, auf die der Programmierer ohne eigenen Programmieraufwand direkt zugreifen kann.

- **Binärprogramm:** Ein direkt ausführbares Programm, z.B. in einer .EXE-Datei unter Windows. Eingeschränkt gilt dies auch für Java-.class-Dateien, die jedoch auf die Java-Laufzeitumgebung angewiesen sind. Binärprogramme werden von einem Compiler aus einem Quellcode erzeugt.

- **.class-Dateien:** Dateien mit der Endung .class enthalten einen fertig kompilierten Javaprogrammteil (eine Klasse). Mehrere .class-Dateien werden sehr häufig zu .JAR-Dateien zusammengefasst. Dies erhöht die Übersichtlichkeit und spart Speicherplatz.

- **Client:** Ein Programm, das sich mit dem Server (Emulator-Kern) verbindet. Hierbei kann es sich z.B. um die Konsole, eine Applikation oder ein Hardware-Interface handeln.

- **Compiler:** Ein Programm, das ein von Menschen Programm im Quellcode in ein von einem Computer ausführbares Binärprogramm umwandelt.

- **core:** siehe Emulator-Kern

- **Counter:** siehe Zähler.

- **Dualzahl:** Eine Ganzzahl, die zur Basis 2 ausschließlich mit 0 und 1 dargestellt wird. Siehe auch BCD-Zahl.

- **Eingangsbyte:** Eine Gruppe von 8 aufeinanderfolgenden liegenden Eingängen. So bilden die Eingänge E0.0 bis E0.7 das Eingangsbyte EB0, die Eingänge E1.0 bis E1.7 das Eingangsbyte EB1 und so weiter.

- **Eingangswort:** Zwei aufeinanderfolgende Eingangsbytes. Das Eingangswort EW0 besteht z.B. aus den beiden Eingangsbytes EB0 und EB1.

- **Emulator:** Ein Emulator ist ein System, welches ein anderes System Original getreu simuliert und dabei auch Detailfragen nicht außer Acht lässt.

- **Emulator-Kern (core):** Der Hauptteil der von mir entwickelten Software, welcher die eigentlich SPS-Emulation durchführt und von Applikationen (Clients) gesteuert wird. Er fungiert als Netzwert-Server.

- **Eventhandler:** Programmfunktion bzw. Java-Methode, die bei bestimmten Ereignissen aufgerufen wird, um ein Programm über dieses Ereignis zu informieren.

- **Fließkommazahl:** Eine Zahl mit Nachkommastellen (diese können jedoch auch 0 sein).

- **Framework:** Ein Programmskelett, dass die Grundfunktionen eines Programms vorgefertigt bereitstellt, und von einem Programmierer für eine konkrete Anwendung erweitert und angepasst werden kann.

- **Ganzzahl:** Eine Zahl ohne Nachkommastellen. Gegenteil zur Fließkommazahl.

- **Gleitkommazahl:** siehe Fließkommazahl

- (Hardware-)**Interface:** Eine elektronische Baugruppe,
    - mit der elektrische Signale für einen Computer auswertbar gemacht werden können.

    - die über Ausgänge verfügt, mit der Computer elektrische Signalpegel (digital oder analog) erzeugen kann.

- **Instruktion:** Eine Anweisung in einem Programm, z.B. in einem Java- oder AWL-Quelltext.

- **.JAR-Dateien:** JAR-Dateien sind fertig kompilierte und zu Paketen gebündelte Java-Programme. Java-Programme in JAR-Paketen können mit dem Programm java.exe aus dem JRE ausgeführt werden. Dazu ist der Befehl
  `java -jar dateiname.jar`
  einzugeben.

- **Java:** Eine Programmiersprache der Firma Sun, die sich besonders dadurch auszeichnet, dass die erzeugten Programme direkt, d.h. ohne zusätzliche Kompilierung, auf einer Vielzahl von Betriebsystemen und Hardwareplattformen lauffähig sind. Diese Flexibilität wird jedoch mit Performance-Nachteilen erkauft, so dass sich Java für zeitkritische Anwendungen nur bedingt eignet. Da das Entwicklungssystem kostenlos verfügbar ist und inzwischen auch Java-Systeme von anderen Herstellern existieren (z.T. auch Open Source), bietet sich Java als Werkzeug in der offenen Lehre an. Weiter Informationen sind unter http://java.sun.com/ erhältlich.

- **Java Development Kit (JDK):** Das JDK ist ein Einwicklungspaket für die Programmiersprache Java. Es richtet sich an Programmierer und enthält unter anderem einen Java-Compiler (javac), das Java-Runtime-Enviroment (JRE) und andere nützliche Werkzeuge, die benötigt werden, um selbst Java-Programme zu entwickeln. Zum einfachen Ausführen von bereits fertigen Java-Programmen kann auf das deutlich kompaktere Java-Runtime-Enviroment (JRE) zurückgegriffen werden.

- **Java-Runtime-Enviroment (JRE):** Das JRE ist die Laufzeitumgebung der Programmiersprache Java. Für die Verwendung dieses Emulators wird mindestens die Version 1.5.0 benötigt.

- **JDK:** siehe Java Development Kit

- **JRE:** siehe Java-Runtime-Enviroment

- **Klasse:** Eine Klasse ist eine Bündel aus Programmcode und Daten, die zur logischen Strukturierung und Erhöhung der Modularität zu einer Einheit verschweißt wurden.

- **Kompiler:** siehe Compiler

- **Konsole:** Eine spezielle Applikation zur Steuerung der virtuellen SPS. Über die Konsole können u.a. Eingänge gesetzt, Ausgänge betrachtet, AWL-Programme in die SPS geladen und die Betriebsart der SPS einstellt werden.

- **Laufzeitumgebung:** Die Laufzeitumgebung bezeichnet verschiedene Softwarekomponenten, die ein Programm zur Ausführung benötigt. Die Laufzeitumgebung stellt dem Hauptprogramm Dienste zur Verfügung, z. B. Bibliotheken mit zusätzlichem Programmcode oder eine abstraktere Sicht auf die

Hardware des Computers. Dieser Emulator stellt gewissermaßen eine Laufzeitumgebung für AWL-Programme dar und läuft seinerseits in der Laufzeitumgebung der Programmiersprache Java, dem JRE.

- **Mehrpunkt-Kommunikation:** siehe Multicast

- **Multicast:** Eine Methode zur Netzwerkkommunikation, bei der mehrere Stationen miteinander kommunizieren. Ein Sender sendet dabei nicht wie bei der Punkt-zu-Punkt-Übertragung (Unicast) an genau einen Empfänger, sondern an eine Gruppe von Empfängern.

- **PDA:** Ein kleiner tragbarer Computer, der normalerweise als Organizer für Termine, Adressen und Aufgaben genutzt wird, aber auch darüber hinausgehende Aufgaben erledigen kann.

- **Port:** Ein einzelner oder eine Gruppe von Ein- oder Ausgängen.

- **Punkt-zu-Punkt-Kommunikation:** siehe Unicast

- **Quellcode/-text:** Ein Programm in einer menschenlesbaren Programmiersprache, z.B. Java oder AWL. Der Quellcode wird durch den Compiler in ein ausführbares, aber dafür für Menschen unlesbares Binärprogramm übersetzt. Java-Quellcode wird in Dateien mit der Endung .java, AWL-Quellcode in Dateien mit der Endung .awl gespeichert.

- **Server:** Ein Programm, welches anderen Programmen (Clients) Dienste über ein Netzwerk bereitstellt. Client und Server können auch auf demselben Computer laufen. Im Rahmen dieser Diplomarbeit bezieht sich das Wort „Server“ in der Regel auf den Emulator-Kern.

- **SFC:** SFCs sind Systemfunktionen, die speicherprogrammierbare Steuerungen in Form einer Bibliothek bereitstellen.

- **Speicherprogrammierbare Steuerung (SPS):** Unter speicherprogrammierbaren Steuerungen werden Minicomputer verstanden, die Steuerungsaufgaben in Industrieanlagen wahrnehmen. Sie verfügen elektrische Ein- und Ausgänge, die über ein Steuerungsprogramm zueinander in Bezug gesetzt werden. SPS sind leistungsfähiger und flexibler als festverdrahtete Schaltungen. Neben logischen Funktionen wie UND, ODER, NICHT bieten sie auch Rechenfunktionen, Zähler, Zeitgeber und komplexe Funktionen wie die Vernetzung mehrerer SPS und Computer über Feldbusse.

- **SPS:** siehe Speicherprogrammierbare Steuerung

- **Suchpfad:** Der Suchpfad ist eine Liste von Verzeichnissen, in denen das Betriebsystem nach ausführbaren Programmen sucht, wenn ein Programm ohne die Angabe des vollständigen Pfades gestartet werden soll. Der Suchpfad wird unter Windows und Linux über die Umgebungsvariable PATH gesetzt. Siehe hierzu das Kapitel zur Installation des Emulators.

- **Symboltabelle:** Die Symboltabelle ist ein Verzeichnis aller Variablen, Konstanten und Parameter eines Bausteins. Die Symboltabelle wird beim Kompilieren eines Bausteins automatisch aus den im Quelltext vorhandenen Variablen- und Parameterdeklarationen erzeugt. Ein dedizierter Symboltabelleneditor ist daher nicht erforderlich.

- **Timer:** siehe Zeitgeber.

- **Treiber:** Eine spezielle Applikation zur Steuerung realer Hardware über Interfaces.

- **Unicast:** Eine Methode zur Netzwerkkommunikation, bei der genau zwei Stationen miteinander kommunizieren (Punkt-zu-Punkt-Verbindung). Siehe auch Multicast.

- **Zähler:** Ein Zähler ist eine Funktion einer SPS, die es ermöglicht, auf eine einfache Art und Weise externe Ereignisse zu zählen. Zähler können vorwärts und rückwärts zählen, auf Null zurückgesetzt und mit beliebigen Startwerten initialisiert werden.

- **Zeitgeber:** Ein Zeitgeber ist eine Uhr innerhalb einer SPS. Zeitgeber ermöglichen z.B. Blinkeffekte bei Meldeleuchten, Ein- und Ausschaltverzögerungen und die Umwandlung von Impulsen.

[EOF]