Diplomarbeit

MES - Manufacturing Execution Systems

Modellierung und Simulation einer Fertigungszelle in der diskreten Fertigung mit Hilfe einer MES-Software

ausgeführt zum Zweck der Erlangung des akademischen Grades
eines Diplom-Ingenieurs

am

Institut für Fertigungstechnik
der Fakultät für Maschinenwesen u. Betriebswissenschaften
der Technischen Universität Wien

unter der Leitung von
a.o.Univ.Prof. Dipl.-Ing. Dr.techn. Burkhard Kittl

durch

Stefan Auer	Johann Weidenauer
Matrikel Nr.: 0125507	Matrikel Nr.: 0125582
Hinterleiten 21	Moniholz 50
3263 Randegg	3524 Grainbrunn

Wien, Oktober 2007

Danksagung

Wir bedanken uns an dieser Stelle recht herzlich bei a.o.Univ.Prof. Dipl.-Ing. Dr.techn. Burkhard Kittl, der es uns ermöglicht hat, unsere Diplomarbeit am Institut für Fertigungstechnik an der Technischen Universität Wien zu schreiben. Er hat uns jederzeit unterstützt und stand uns immer mit Rat und Tat zur Seite. Er war für uns nicht nur der Betreuer unserer Diplomarbeit, sondern auch ein Mensch, mit dem es eine Freude war, zusammenzuarbeiten. Ein herzliches Dankeschön auch an alle Freunde und Studienkollegen, die uns mit fachlichen Informationen sowie wertvollen Anregungen unterstützt haben. Wir hatten eine schöne Zeit am Institut für Fertigungstechnik und bedanken uns deshalb auch bei allen Mitarbeitern, die uns sehr freundlich aufgenommen haben.

Mein spezieller Dank gilt meinen Eltern, die mir das Studium an der TU Wien ermöglicht haben. Weiters bedanke ich mich bei allen Freunden, Familienmitgliedern und speziell bei meiner Freundin für ihre Geduld und den Rückhalt, besonders am Ende meiner Studienzeit.

Stefan Auer

Ich möchte mich ganz besonders bei meinen Eltern bedanken, die mir mein Technikstudium in Wien ermöglicht haben. Auch meiner ganzen Familie sowie meinen Freunden gilt dieser Dank, die mir in schwierigen Situationen den nötigen Rückhalt gegeben haben. Besonders danken möchte ich auch meiner Freundin, die mich auf diesem Lebensabschnitt begleitet hat und mir immer hilfreich zur Seite stand.

Johann Weidenauer

Kurzfassung

Diese Diplomarbeit beschäftigt sich mit dem Einsatz von Manufacturing Execution Systems (Fertigungsleitsystemen). Speziell zielt sie aber auf die diskrete Fertigung ab. Da dieses Thema zur Zeit in der Wirtschaft sehr aktuell ist, wird am Institut für Fertigungstechnik an der Technischen Universität Wien eine Versuchsanlage aufgebaut, mit der die Abläufe in einer Fertigungszelle nicht nur simuliert, sondern auch praktisch durchgeführt werden können. Dies ist ein größeres Projekt des Institutes, das später mehrere Diplomarbeiten umfassen soll. Diese Diplomarbeit stellt eine Basis des Projektes dar und beschäftigt sich mit der Modellierung und Simulation einer Fertigungszelle in der diskreten Fertigung mit Hilfe einer MES-Software.

Zu Beginn der Diplomarbeit wird eine Einführung über das Thema Manufacturing Execution Systems gegeben. In weiterer Folge wird dann das Szenario an der Versuchsanlage behandelt. Es soll gezeigt werden, wie eine Steuerung durch ein MES aussehen kann, wie die Verbindung zwischen der Unternehmensleitebene und der Fertigungsebene durch ein Manufacturing Execution System hergestellt wird und wie die Fertigung dann tatsächlich abläuft. Dazu müssen alle Anlagenprozesse genau definiert und eine benutzerfreundliche Bedienoberfläche erstellt werden.

Da die Fertigungszelle gerade aufgebaut wird und somit physisch nicht zur Verfügung steht wird ein Simulator entwickelt, der alle Eigenschaften der realen Anlage besitzt. Dieser Simulator soll auch eine Visualisierung ermöglichen. Die bei der Diplomarbeit eingesetzte MES-Software „SIMATIC IT Production Suite Version 6“ ist ein Produkt der Firma Siemens.

Abstract

This thesis is based upon the use of Manufacturing Execution Systems (MES) with specific focus on the discrete parts manufacturing, which is a central issue in today’s industry. In response to this the Institute of Production Engineering of the Vienna University of Technology is building up a test facility to simulate different case scenarios. It is a significant project carried out by the Institute and will consist of a range of thesises and project works to be carried out. This diploma thesis is the first part of the overall project and it contains the modelling and simulation of a manufacturing cell in the discrete parts manufacturing with the use of a MES.

The first section of the thesis gives a short overview concerning the theory of Manufacturing Execution Systems. The next part describes the scenario of the test facility. It should show how a Manufacturing Execution System can be used to control the manufacturing cell and how it realises the link between the company management level and the production level.

Inhaltsverzeichnis

Abbildungsverzeichnis

Abkürzungsverzeichnis

AG	Arbeitsgang
ANSI	American National Standards Institute
ASP	Active Server Pages
CAB	Client Application Builder
CIM	Computer Integrated Manufacturing
CNC	Computerized Numerical Control
COM	Component Objekt Model
DIS	Data Integration Service
DLL	Dynamic Link Library
ERP	Enterprise Resource Planning
GSI	Global Settings Interface
HTML	Hypertext Markup Language
HUT	Handling Unit
ID	Identifizierung
ISA	Instrumentation Systems and Automation Society
ISO	International Organization for Standardization
KVP	Kontinuierlicher Verbesserungsprozess
MES	Manufacturing Execution Systems
MESA	Manufacturing Enterprise Solution Association
MM	Material Manager
MS	Microsoft
PC	Personal Computer
PM	Production Modeler

POM	Production Order Manager
PPS	Produktionsplanung und – steuerung
SCM	Supply Chain Management
SPS	Speicherprogrammierbare Steuerung
SQB	Smart Query Builder
SQL	Structured Query Language
UML	Unified Modeling Language
VDI	Verein Deutscher Ingenieure
XML	Extensible Markup Language

Vorwort

Fertigungsunternehmen (z.B. Automobilzulieferindustrie) sind sehr bemüht, ihre Fertigung so wirtschaftlich wie möglich zu gestalten. Dazu muss man die jeweiligen Fertigungsprozesse so schnell und flexibel wie möglich handhaben und weiters die Möglichkeit zu einer zeitnahen Optimierung bieten. Klassische ERP-Systeme sind normalerweise nicht in der Lage diese Aufgaben zu übernehmen. Dies ist darin begründet, dass normales ERP nicht direkt mit der Fertigungsebene gekoppelt ist und deshalb Daten nicht in Echtzeit erfasst, beziehungsweise verarbeitet werden können. Es wird hier meist mit nicht detaillierten Daten gearbeitet und daher lassen sich nur grobe Aussagen über die realen Produktionsabläufe treffen.

Viele Unternehmen versuchen im Moment eine direkte Anbindung ihrer Fertigungsebene (Fertigungs-/ Produktionsprozess) an die Unternehmensleitebene (ERP) herzustellen. Die Spalte zwischen diesen beiden Ebenen soll durch Fertigungsmanagementsysteme, so genannte Manufacturing Execution Systems (MES), geschlossen werden. MES-Systeme bilden Fertigungsprozesse zeitnah ab um die Planung und Steuerung der jeweiligen Prozesse so effizient und transparent wie möglich zu gestalten. Dazu soll man sich mit Hilfe von MES jederzeit über den detaillierten, aktuellen Zustand der Fertigung informieren können, das System muss auch Rückmeldungen an die betreffenden Sachbearbeiter weiterleiten können.

Der zielgerichtete Einsatz von MES kann die Wertschöpfung einer Fertigung immens steigern.

1 Aufgabenstellung

1.1 Einleitung

Am Institut für Fertigungstechnik an der Technischen Universität Wien läuft derzeit ein größeres Projekt zum Fachgebiet Manufacturing Execution Systems (MES). Derzeit werden am Markt verschiedene Softwarelösungen angeboten. Es gibt Komplettsysteme, die alle Komponenten eines MES bereitstellen und auch Module, die nur einzelne MES-Bereiche abdecken.

Am Institut soll eine Demonstrationsanlage aufgebaut werden. Diese Anlage wird verwendet, um verschiedene Szenarien über den Einsatz von Manufacturing Execution Systems und deren Schnittstellen zu anderen Systemen zu erproben. Ein Hauptaugenmerk wird dabei auf den Einsatz von MES in der diskreten Fertigung gelegt.

Die Demoanlage soll sowohl dem interessierten Fachpublikum als auch den Studenten der TU Wien in den entsprechenden Vertiefungslehrveranstaltungen zur Verfügung stehen. Dazu sind die realisierten Konzepte, Strukturen und Abläufe in geeigneter Weise medial aufzubereiten. Die zu schaffende Anlage soll auch als Testfeld für gemeinsame Entwicklungen mit industriellen Partnern (Software, Schnittstellen etc.) dienen.

Diese Diplomarbeit bildet die Basis des übergeordneten Projekts. Auf ihr sollen später noch weitere Diplom- bzw. Projektarbeiten aufsetzten. Sie soll zeigen, welche Möglichkeiten sich durch den Einsatz von Manufacturing Execution Systems (Fertigungsleitsysteme) ergeben.

1.2 Aufgabenstellung für diese Diplomarbeit

Im Labor des Instituts für Fertigungstechnik wird derzeit eine Fertigungszelle aufgebaut. Diese Zelle soll durch ein Manufacturing Execution System gesteuert werden. Die Aufgabe dieser Diplomarbeit ist es, mit einer (am Markt erhältlichen) MES-Software die Steuerung zu realisieren, dabei sollen Fertigungsaufträge aus einer Enterprise Ressource Planning Lösung übernommen werden. Diese Aufträge sollen durchgeführt und nach deren Fertigstellung wieder in das ERP zurückgemeldet werden.

1.2.1 Definition der einzelnen Aufgaben

- Definieren der einzelnen Prozesse, die in der Anlage ablaufen, wobei alle Prozessschritte zu erarbeiten sind. Begonnen wird mit dem Freigeben des Fertigungsauftrages durch ein ERP-Programm bis hin zum Zurückmelden eines fertigen Auftrages.

- Um ein Manufacturing Execution System einsetzen zu können, muss die gesamte Anlage im System modelliert werden. Dies umfasst alle vorhandenen Anlagenkomponenten, deren Eigenschaften und alle in der Anlage ablaufenden Prozesse.
- Herstellen der Kommunikation zwischen MES und Fertigungszelle bzw. MES und ERP-Software/Scheduler. Damit im MES Aufträge und Arbeitsgänge übernommen und zurückgemeldet werden können, muss eine Kommunikation zwischen MES und ERP ermöglicht werden. In weiterer Folge muss das MES mit der Anlagensteuerung kommunizieren um die Führung zu gewährleisten.
- Eine benutzerfreundliche Bedieneroberfläche zum Verwalten der Aufträge, der Arbeitsgänge und zum Steuern der Anlage soll erstellt werden.
- Erstellen eines Simulators für die Anlage, damit auch mit dem MES gearbeitet werden kann, solange die reale Anlage aufgebaut wird, zu Vorführungszwecken oder wenn die Anlage nicht zur Verfügung steht.

1.2.2 Wahl der Manufacturing Execution System Software

Nach einer Analyse der am Markt erhältlichen MES-Softwarelösungen ist die Entscheidung auf die SIMATIC IT Production Suite (Version 6, Service Pack 1) von Siemens gefallen.

Dieses Programm hat einen modularen Aufbau und basiert auf einem fortschrittlichen Ansatz. Die SIMATIC IT Production Suite orientiert sich außerdem an der ISA/ANSI 95 Norm. Diese ist derzeit die wichtigste Norm auf dem Gebiet Manufacturing Execution Systems.

SIMATIC IT hat eine Hauptkomponente, den Production Modeler, in der man die Anlage sowie die ablaufenden Prozesse modelliert. Der Production Modeler ist also nicht nur eine Modellierungsumgebung, sondern dient auch als Process-Engine. Weiters bietet das SIMATIC IT beispielsweise noch Komponenten zur Auftrags-, Personal- und Materialverwaltung. Sehr gute Möglichkeiten bestehen auch zur Kommunikation zwischen den einzelnen Systemkomponenten und zu Softwarepaketen von Drittanbietern, was für dieses Projekt sehr wichtig ist. Dazu bietet SIMATIC IT die Komponente Data Integration Service (DIS) sowie auch eine Standard-COM-Schnittstelle an.

Obwohl die Auftragsfeinplanung Teil eines vollständigen MES ist wird diese Komponente in der SIMATIC IT Production Suite nicht bereitgestellt. Sie wird deshalb durch ein Feinplanungstool (Scheduler) übernommen. Die Bedienung dieses Schedulers ist aber nicht Gegenstand dieser Diplomarbeit und deshalb nicht näher beschrieben.

1.2.3 Projektabgrenzung

Da diese Diplomarbeit von zwei Diplomanden erstellt wird, ist es notwendig, eine genaue Abgrenzung zwischen den Themengebieten der Diplomarbeit anzugeben. Der Allgemeine Teil wird von beiden Diplomanden (Stefan Auer, Johann Weidenauer) behandelt. Später teilen sich dann die Aufgabenbereiche wie folgt auf:

- Stefan Auer widmet sich der Bedienung der SIMATIC IT Production Suite. Das umfasst das Modellieren der Anlage mit allen Anlagenkomponenten und -prozessen und die Schnittstelle zwischen MES und Scheduler bzw. ERP. In weiterer Folge gehört auch das Auftrags- und Materialmanagement in den jeweiligen SIMATIC IT Komponenten zum Aufgabengebiet.
- Der Aufgabenbereich von Johann Weidenauer umfasst das Programmieren der Benutzeroberfläche, des COM-Wrappers und des Simulators. Der COM-Wrapper ermöglicht die Kommunikation mit der Anlage, der Simulator ein Arbeiten mit dem Manufacturing Execution System, wenn die physische Anlage nicht zur Verfügung steht.

1.2.4 Ziele der Diplomarbeit

Diese Diplomarbeit soll die Basis für weiterführende Forschungsarbeiten zum Thema MES bilden. Es soll ein Grundaufbau erstellt werden, der die wesentlichen Funktionen eines MES anwendet und spätere Erweiterungen ermöglicht.

Im Rahmen der Diplomarbeit sollen Erfahrungen über den Einsatz von Manufacturing Execution Systems zur Fertigungssteuerung gesammelt werden.

2 Allgemeines zu MES

Dem allgemeinen Teil über Manufacturing Execution Systems liegen in erster Linie folgende Quellen zugrunde:

- Kletti, Jürgen: MES – Manufacturing Execution Systems/Moderne Informationstechnologie zur Prozessfähigkeit der Wertschöpfung, Heidelberg, 2006.
- VDI: VDI5600/Manufacturing Execution Systems/Fertigungsmanagementsysteme, Düsseldorf, 2006.

2.1 Historische Entwicklung von MES

Die Grundidee und die Wurzeln sind in Werkzeugen aus den 80er Jahren verankert. In dieser Zeit wurde mit dem Aufkommen des CIM-Konzeptes (Computer Integrated Manufacturing) begonnen, die Bereiche Fertigungsplanung, Personal und Qualitätssicherung an IT-Systeme anzubinden. Die breite Durchsetzung von CIM wurde letztlich verhindert, da keine betriebswirtschaftlichen Systeme wie PPS oder ERP mit der erforderlichen Leistungsfähigkeit vorhanden waren.

In den 90er Jahren wurden Lösungen für die Bereiche PPS, Betriebs- und Maschinendatenerfassung, Fertigungsleitstände, Qualitätssicherungssysteme, Materialwirtschaft sowie Personalmanagement verwendet. Die Systeme waren jedoch voneinander unabhängig und hatten keine standardisierten Schnittstellen. Ende der 90er Jahre haben die Organisationen MESA und ISA mit Standardisierungsaktivitäten begonnen. Dabei stand die Integration aller Bereiche im Vordergrund. Ist diese Integration vorhanden, kann man bereits von einem Manufacturing Execution System sprechen.

Im deutschsprachigen Raum hat der Verein Deutscher Ingenieure (VDI) die Initiative ergriffen und 2003 damit begonnen, eine Richtlinie über das Fachgebiet MES zu erstellen.

2.2 Definition laut Entwurf der VDI Richtlinie 5600

Die Summe und Komplexität der benötigten Informationen sowie die Notwendigkeit der daraus folgenden sehr zeitnahen Reaktionen erfordern ein prozessnah operierendes Fertigungsmanagementsystem – ein MES (Manufacturing Execution System).

Ein MES ist ein umfassender Treiber für die Organisation und Durchführung des Produktionsprozesses. Dazu hat es umfassende Aufgabenbereiche abzuwickeln und/oder zu unterstützen:

- Organisation und Unterstützung aller erforderlichen Aktivitäten im Produktionsprozess
  - Organisation und Unterstützung von Prozessvorgaben (Auftragsbearbeitung in Zeit und Reihenfolge, Einsatz von Personal, Ressourcen und Material, Vorgabe der Qualitätsförderungen usw.)
  - Organisation und Unterstützung aller operativen Aktionen (Sicherung der Ressourcenverfügbarkeit und Ressourcenbereitstellung, Sicherung des Materialflusses, Sicherung der Produkt- und Prozessqualität usw.)
  - Organisation und Unterstützung der Analyse von Produktionsaktivitäten zur Nachverfolgung und zur Erschließung von Verbesserungspotenzial (Berechnung von Kennziffern, Input für KVP)

Mit seiner Funktionalität muss es die drei zeitlichen Aspekte der Prozessdurchführung abdecken:

Prognostischer Aspekt → Produktionsprozessplanung

Aktueller Aspekt → Produktionsprozessregelung und –steuerung

Historischer Aspekt → Produktionsprozessaus und –bewertung

- Realisierung vom Wirkungskreislauf aller Aktionen zur Durchführung des Produktionsprozesses, darin enthalten der
- Austausch von Informationen mit der Umgebung, bezogen auf die
  - Unternehmensleitebene (ERP) sowie benachbarte betriebsunterstützende Systeme, z.B. Product and Process Engineering (P/PE), Supply Chain Management (SCM), und die
  - Fertigungsebene mit den Fertigungs- und Produktionsprozessen.

Abbildung 2.1 zeigt das grundsätzliche MES-Konzept sowie seine Einbindung in die Leitsysteme eines Unternehmens. Es werden acht MES-Aufgaben definiert, deren Zusammenspiel die geschilderte umfassende Unterstützung des Fertigungs-/Produktionsprozesses erlaubt. Eine konkrete Ausprägung eines MES muss jedoch nicht die Realisierung aller acht Aufgaben umfassen, sondern wird im Leistungsumfang jeweils den Anforderungen des Anwenders angepasst.

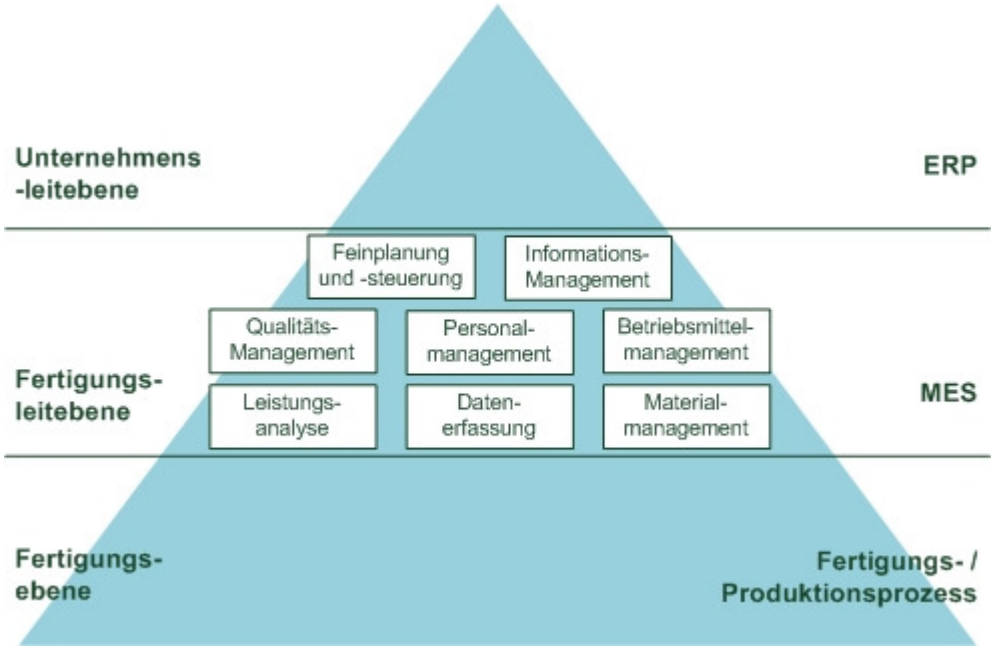

Abbildung 2.1: Einordnung von MES in den Leitebenen eines Unternehmens[1]

2.3 Möglicher Nutzen eines MES

Durch den Einsatz eines MES kann ein Unternehmen folgende Verbesserungen erreichen:

- Eine vorausschauende Beherrschung aller Bedingungen im Produktionsprozess sowie eine präventive Beeinflussung dieser Bedingungen.
- In der Arbeitsvorbereitung und Fertigungsorganisation kann das Personal entlastet werden, da Routineaufgaben vom MES übernommen werden.
- Ein MES unterstützt das integrierte Produktionsmanagement im Zusammenspiel von wertschöpfenden Produktionsoperationen, menschlichen Aktivitäten und angebotenen Systemfunktionen.
- Der Automatisierungsgrad der Produktionseinrichtungen kann durch ein MES erhöht werden.
- Möglichkeiten zur Einhaltung der Produkthaftungsvorschriften werden durch ein MES bereitgestellt.

2.4 Normung und Standardisierung

Der Begriff Fertigungsmanagementsysteme beziehungsweise Manufacturing Execution Systems ist sehr weitläufig, deshalb gibt es zurzeit mehrere Bemühungen eine genaue

[1] Quelle: VDI, (2006:4)

Abgrenzung des Begriffs zu bestimmen. Es soll geklärt werden, welche Aufgaben und Bereiche ein MES im Detail übernimmt. Zu diesem Thema gibt es verschiedene Ansichten. Hier werden nur die wichtigsten Normungs- und Standardisierungsbemühungen genannt.

- Die Organisation MESA (Manufacturing Enterprise Solution Association) ist eine Vereinigung von Herstellerfirmen, Lösungsanbietern und Fachleuten aus der Industrie. Diese haben das Ziel, Wissen zum Thema Enterprise Solutions zu liefern. Die Vereinigung beschreibt zwölf Funktionsgruppen, die für die effiziente Unterstützung des Fertigungsmanagements wichtig sind.
- Die Normen ANSI/ISA 95 (Enterprise-Control System Integration) und ISO/IEC 62264 beschäftigen sich ebenfalls mit dem Thema MES.

  Die ANSI/ISA-95 besteht derzeit aus drei Teilen. Diese Teile beschäftigen sich mit folgenden Inhalten:

  1. Der erste Teil beschäftigt sich mit der Definition von Schnittstellen zwischen den Fertigungssteuerungsfunktionen und den einzelnen Unternehmensfunktionen. Das Ziel ist es, Risken, Kosten und Fehler, die mit der Schnittstellenimplementierung zusammenhängen, zu reduzieren[2].
  2. Der Bereich des zweiten Teils ist darauf beschränkt, Attribute für die Objektmodelle aus dem ersten Teil zu definieren[3].
  3. Der dritte Teil definiert Aktivitätenmodelle des Produktionsvorgangs-Managements. Diese Modelle ermöglichen IT Systemen die Systemintegration zu kontrollieren[4].

  Die ANSI/ISA-95 Norm wird ständig erweitert. Derzeit befinden sich zwei neue Teile in Vorbereitung.
- Der VDI (Verein Deutscher Ingenieure) hat die VDI Richtlinie 5600 veröffentlicht. Diese Richtlinie soll Planern in Fertigungsunternehmen und potentiellen MES-Anwendern als Leitfaden dienen und eine fachlich fundierte Hilfestellung bieten.

[2] Vgl. ANSI/ISA-95.00.01-2000, 2000.
[3] Vgl. ANSI/ISA-95.00.02-2001, 2001.
[4] Vgl. ANSI/ISA-95.00.03-2005, 2005.

3 Beschreibung der Demonstrationsanlage

In diesem Kapitel wird das Layout der Anlage näher beschrieben. In weiterer Folge wird erklärt, wie die Systemlandschaft und die Kommunikationsstruktur in der Zelle einerseits und der Zelle übergeordnet andererseits aussehen.

3.1 Anlagenlayout

Die eigentliche Fertigungszelle besteht aus mehreren Objekten:

- Werkstückpuffer mit Be-/Entladestation und Bearbeitungsstation
- Roboter
- Drehzentrum mit Stangenlader
- Reinigungsstation
- Entgratstation
- Zellenrechner
- Bedien-PC

Diese sind räumlich so angeordnet, wie in Abbildung 3.1 zu sehen ist.

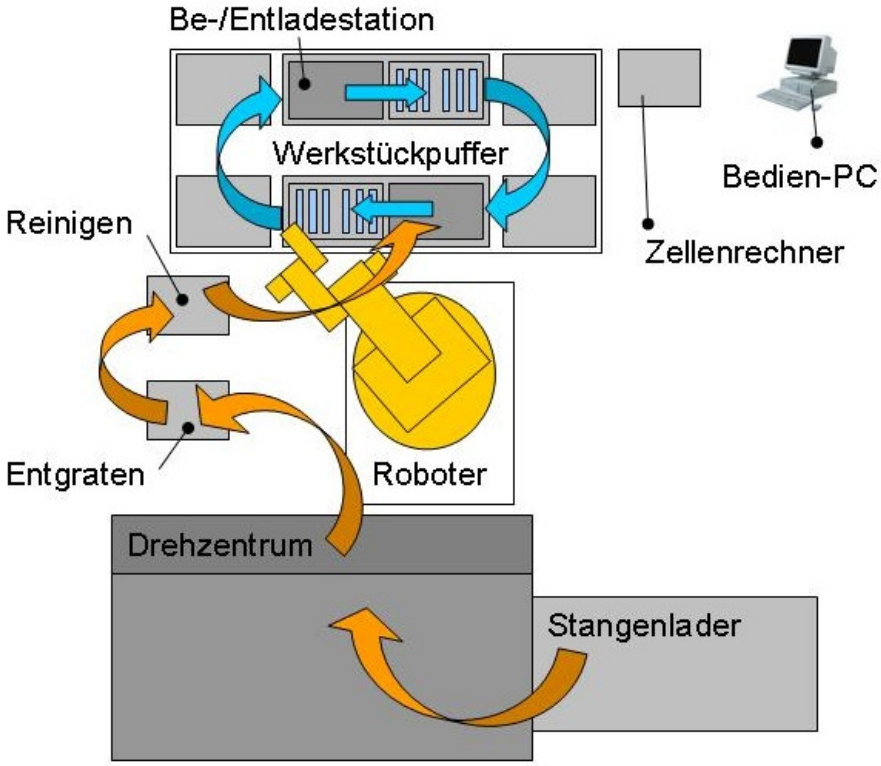

Abbildung 3.1: Anlagenlayout

An der Be-/Entladestation werden leere Paletten durch einen Bediener eingelegt und in einen Rohteilpuffer eingeschleust. Der Palettentyp kann je nach Endprodukt variieren und wird vom MES vorgegeben. Die Leerpaletten laufen danach über den Umsetzer einzeln in die Bearbeitungsstation ein.

Die Maschine wird nun aus dem Stangenlader mit Rohmaterial beschickt und die Bearbeitung durchgeführt. Nach Fertigstellung des Werkstückes wird es von einem Greifer in der Maschine entnommen und abgelegt. Der Roboter entnimmt nun das fertige Werkstück an der Entnahmestelle der Maschine. Weitere Prozessschritte wie Reinigen oder Entgraten werden in Kooperation zwischen Roboter und den entsprechenden Stationen (Reinigungsstation und Entgratstation) durchgeführt. Der Roboter übernimmt also nicht nur das Werkstückhandling, sondern ist auch an der Bearbeitung beteiligt.

Nach Durchführung aller erforderlichen Prozess-Schritte werden die Fertigteile in der Palette auf der Bearbeitungsstation abgelegt.

Wurde die Palette fertig bestückt, so wird diese in den Fertigteilpuffer geschleust. Wenn die Be-/ Entladestation frei ist, wird die Palette über den Umsetzer zu dieser weiter transportiert. Anschließend kann das Bedienpersonal die Teile auf der Palette, sowie die Palette selbst entnehmen.

3.2 Systemlandschaft und Kommunikationsstruktur

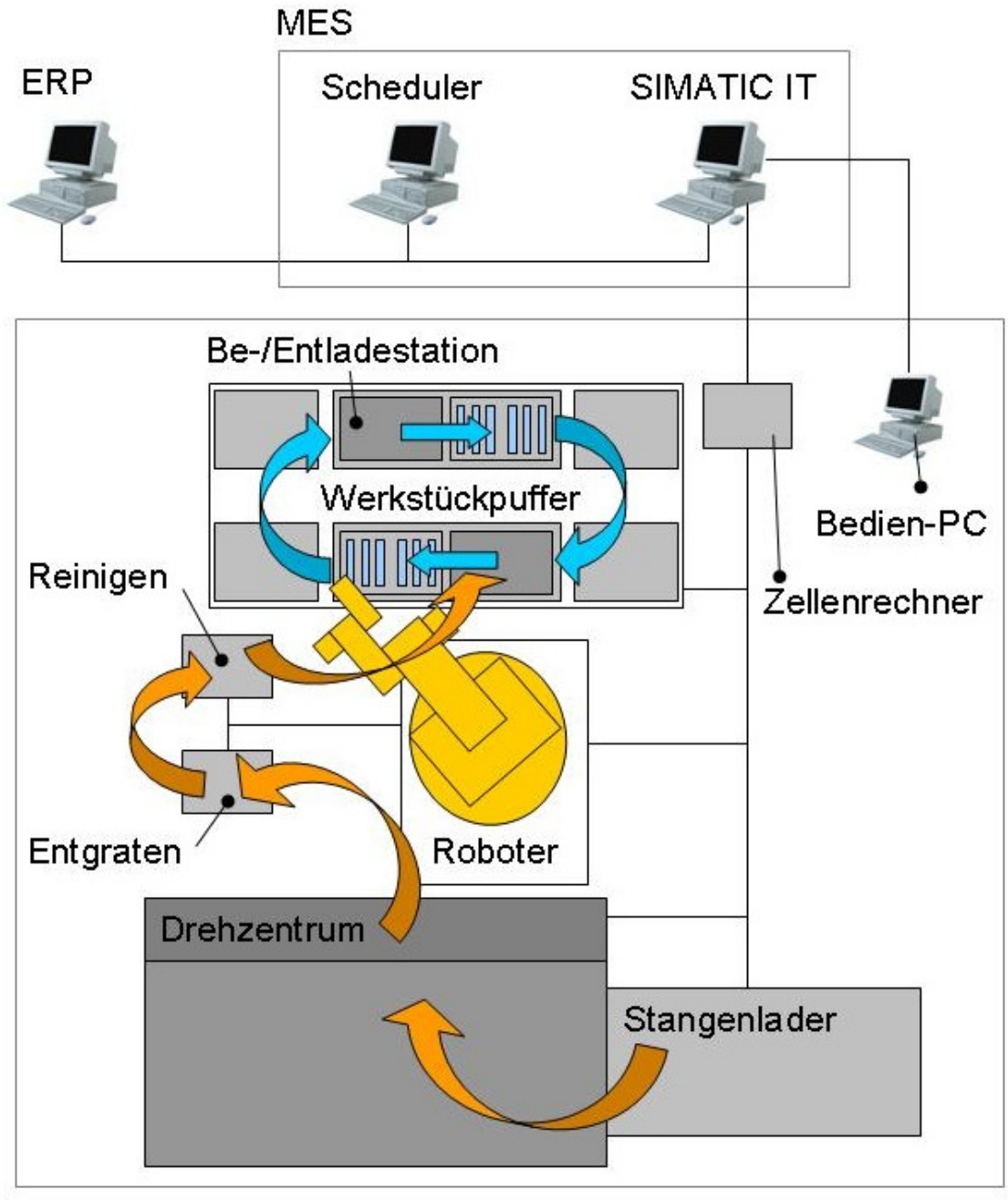

Abbildung 3.2: Systemlandschaft und Kommunikationsstruktur der Anlage

Die obige Abbildung zeigt, zusätzlich zur Fertigungszelle, wie die übergeordneten Komponenten in der Systemlandschaft angeordnet sind. Diese übergeordneten Komponenten sind das ERP-System und das MES System. Das Manufacturing Execution System besteht hier aus einem Feinplanungstool (Scheduler) und SIMATIC IT. Im folgenden Teil wird sehr oft der Ausdruck MES verwendet. Damit ist aber immer die SIMATIC IT Production Suite gemeint.

Freigegebene Fertigungsaufträge werden von einem ERP-System an ein Feinplanungstool übergeben. Dort werden Sie terminiert und an SIMATIC IT übergeben. Fertiggestellte Aufträge und Arbeitsgänge werden mit ihren Ist-Daten an das ERP-System rückgemeldet.

3.2.1 Kommunikation zwischen ERP-System und dem Feinplanungstool

Diese Kommunikation ist nicht Teil der Diplomarbeit und wird deshalb hier nicht näher behandelt.

3.2.2 Kommunikation zwischen Feinplanungstool und MES bzw. MES und ERP

Ein wichtiger Teil dieser Diplomarbeit ist die Kommunikation zwischen Scheduler und MES aber auch zwischen ERP-System und MES.

Feingeplante Fertigungsaufträge werden an das MES übergeben. Fertige Arbeitsgänge und Fertigungsaufträge werden danach in das ERP-System rückgemeldet.

Dieser Datenaustausch basiert auf XML-Dateien, die über die Komponente Data Integration Service (DIS) der SIMATIC IT Production Suite ins MES eingelesen bzw. aus dem MES übergeben werden können. Näheres dazu findet man in Kapitel 8.3.

3.2.3 Kommunikation zwischen MES und der Fertigungszelle

Das MES muss mit der Zelle kommunizieren, um die Anlage steuern zu können.

Auf dem Bedien-PC laufen Anwendungen des MES (Bedienoberfläche für Arbeitsgangverwaltung, …). Diese Anwendungen werden in Visual Basic und dem so genannten Client Application Builder von SIMATIC IT erstellt. Dies ist in Kapitel 9 ausführlich beschrieben.

Das MES ist über den Zellenrechner mit den Anlagekomponenten (Roboter, Palettenpuffer, …) verbunden. Vom MES aus werden Methoden aufgerufen, welche vom Zellenrechner zur Verfügung gestellt werden. Die Kommunikation läuft dabei über eine COM-Schnittstelle. Die Erstellung dieser Methoden ist in Kapitel 10 näher beschrieben.

3.3 Anlagenkomponenten

Im nachfolgenden Teil werden die Komponenten und Abläufe in der Demo-Anlage genauer beschrieben. Diese Anlagenkomponenten werden in der Arbeit durch einen in VB programmierten Simulator abgebildet und ermöglichen so ein Arbeiten mit dem MES-Programm, ohne Zugriff auf die Anlage zu haben.

3.3.1.1 Werkstückpuffer

Der Werkstückpuffer dient im Demoszenario als Speicher für leere Paletten und solche Paletten, die bereits mit Fertigbauteilen befüllt wurden.

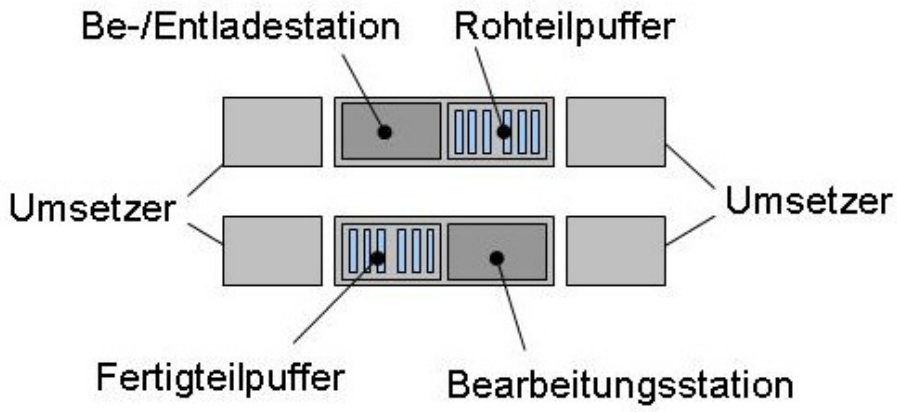

Abbildung 3.3: Layout des Werkstückpuffers

Der Werkstückpuffer besteht aus folgenden Elementen

- Be-/Entladestation: Hier werden leere Paletten eingeschleust. Paletten aus dem Fertigteilpuffer werden entladen, und, wenn nicht mehr benötigt, entnommen. .
- Rohteilpuffer: Von hier läuft die Palette nach Freigabe in die Bearbeitungsstation.
- Umsetzer_1: Nach Freigabe der Palette in der Be-/Entladestation wird sie über die beiden Umsetzer in den Rohteilpuffer transportiert.
- Bearbeitungsstation: Die Palette wird fixiert, damit der Roboter die Fertigteile in der Palette ablegen kann.
- Fertigteilpuffer: Ist die Be-/Entladstation frei, läuft die Palette im Fertigteilpuffer in die Be-/Entladestation.
- Umsetzer_2: Nach Abarbeitung einer Palette läuft diese über weitere Umsetzer in den Fertigteilpuffer.

Bis auf die Umsetzer sind alle Palettenplätze als Hubstationen ausgeführt.

Die vom Puffer auszuführenden Bewegungen sind

- Einschleusen einer Leerpalette von der Be-/Entladestation zum Rohteilpuffer
- Transport der Palette vom Rohteilpuffer über einen Umsetzer zur Bearbeitungsstation
- Einschleusen der bestückten Palette in den Fertigteilpuffer

- Umsetzten der Palette vom Fertigteilpuffer zur Be-/Entladestation

Der Puffer ist mit einer SPS ausgestattet, in der die Abläufe für diese Bewegungen programmiert sind. Angestoßen werden diese Bewegungen durch das MES, das die Aufrufe über den Zellenrechner sendet.

3.3.1.2 Drehzentrum mit Stangenlader, Roboter, Entgrat- und Reinigungstation

Das Drehzentrum wird über einen Stangenlader mit Rohteilen bestückt. Die Steuerung der Maschine ist mit dem Zellenrechner verbunden. Der Anstoß zur Bearbeitung und die Information über das zu fertigende Produkt kommen vom MES

3.3.1.3 Roboter, Entgrat- und Reinigungsstation.

Der Roboter dient zur Manipulation der Werkstücke. Er entnimmt fertige Teile aus dem Drehzentrum, transportiert diese zur Entgrat- und Reinigungsstation und legt das Werkstück schließlich auf der Palette in der Bearbeitungsstation ab.

Die Robotersteuerung ist mit dem Zellenrechner verbunden und übernimmt auch die Ansteuerung der Entgrat- und Reinigungsstation. Das Manufacturing Execution System löst wiederum die Vorgänge aus.

3.3.1.4 Zellenrechner

Der Zellenrechner stellt die Verbindung zwischen MES und den Anlagenkomponenten her. Das MES sendet COM-Aufrufe, die Funktionen der Anlage auszulösen.

Folgende Anlagenfunktionen werden über COM-Aufrufe des MES angestoßen und durch den Zellenrechner an die Steuerungen der Anlagenkomponenten weitergeleitet:

- Palette in Rohteilpuffer einschleusen
- Palette zur Bearbeitungsstation transportieren
- Werkstück bearbeiten und auf Palette ablegen
- Palette in den Fertigteilpuffer einschleusen
- Palette zur Be-/Entladestation transportieren

3.3.1.5 Bedien-PC

Der Bedien-PC hat folgende Aufgaben:

- Gewährleisten der Arbeitsgangverwaltung für die Fertigungszelle
- Bedienoberfläche für die Be- und Entladung von Paletten entsprechend dem Arbeitsgangvorrat (MES Funktionalität)
- Der Bedien-PC bietet auch die Möglichkeit eines Handbetriebes, welcher über die Benutzeroberfläche bedient wird.

Der Bedien-PC ist direkt an das MES angebunden, welches Anwendungen auf dem PC zur Verfügung stellt.

3.4 Simulator

Im oberen Teil dieses Kapitels wurden die reale Anlage und ihre Komponenten beschrieben bzw. definiert. Dieser Teil der Arbeit beschäftigt sich mit der Modellierung und Simulation einer Fertigungszelle. Die reale Anlage wird in dieser Arbeit durch einen Simulator mit einer Anlagenvisualisierung abgebildet. Diese Simulation läuft auf dem Bedien-PC, welcher also neben seinen eigentlichen Aufgaben auch als Simulationstool dient, indem er Signale des MES (COM-Aufrufe) anzeigt, Abläufe der Anlage simuliert, Bedienereingaben ermöglicht und die erforderlichen Signale (Rückmeldungen) für das MES erzeugt.

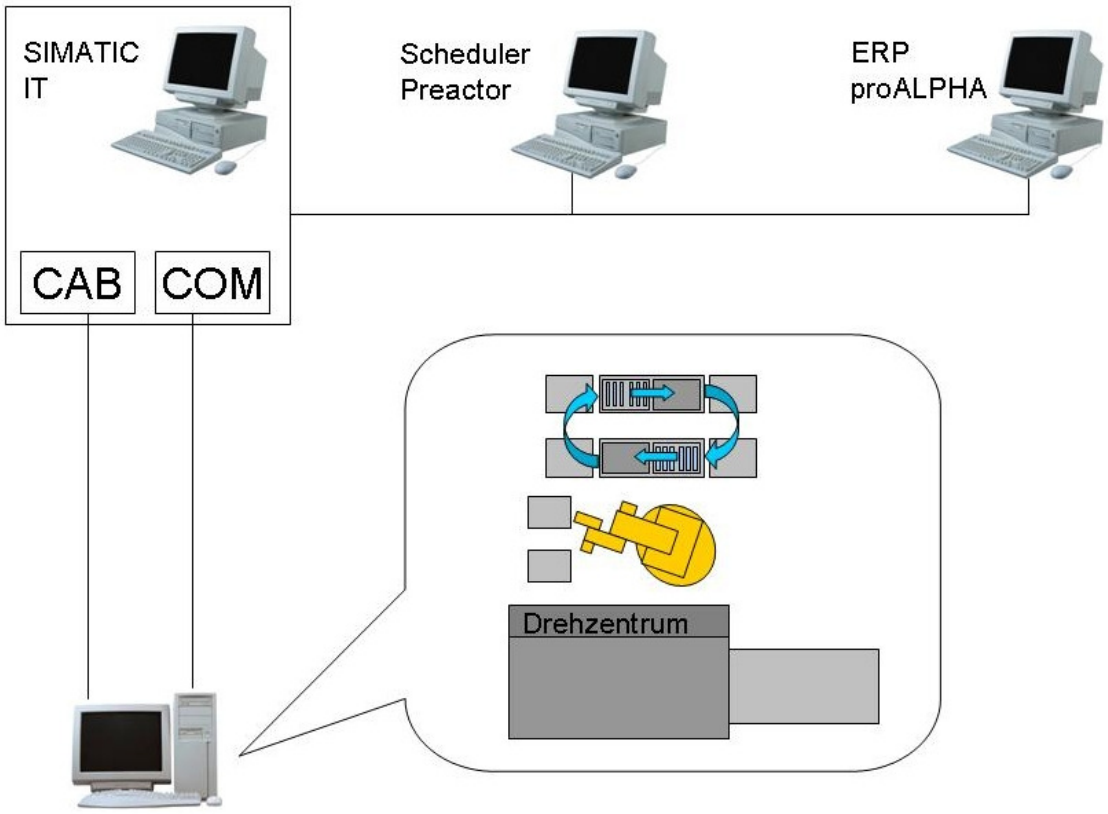

Abbildung 3.4: Simulator

Die Anlage soll dabei grafisch dargestellt und in einfacher Art und Weise animiert werden. Auf diese Weise können die MES-Prozesse auch ohne Verfügbarkeit der Anlage entwickelt und getestet werden.

4 Ablaufbeschreibung

In diesem Kapitel werden die genauen Abläufe bei der Abwicklung eines Fertigungsauftrages beschrieben. Mit dem Start erfolgt die Freigabe des Auftrages durch das ERP und zum Schluss wird der Fertigungsauftrag an das ERP rückgemeldet. Die folgenden Abbildungen sind als UML 2.1 Aktivitätendiagramme ausgeführt und zeigen den Auftrags- und Arbeitsgangablauf vom Ausschreiben des Auftrages bis zu dessen Fertigstellung.

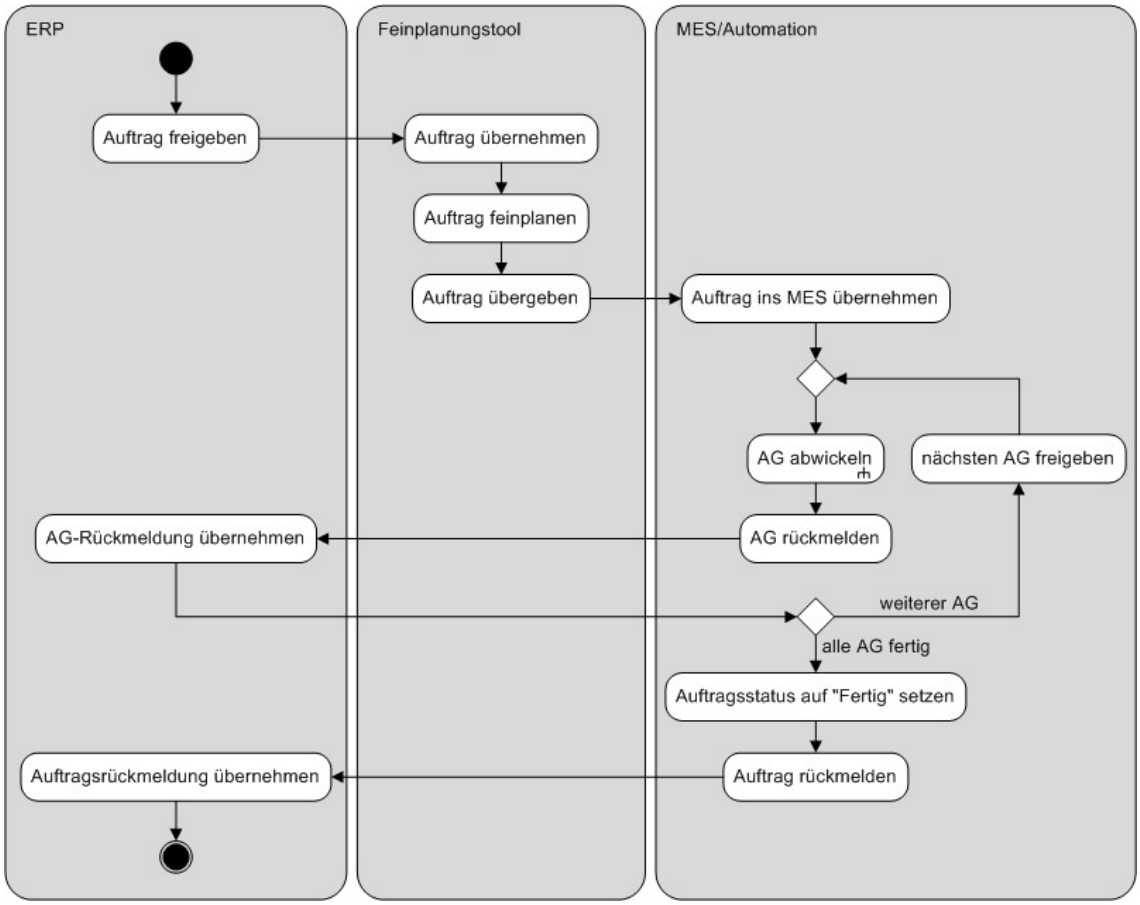

Abbildung 4.1: UML-Aktivitätendiagramm für die Abwicklung eines Auftrages

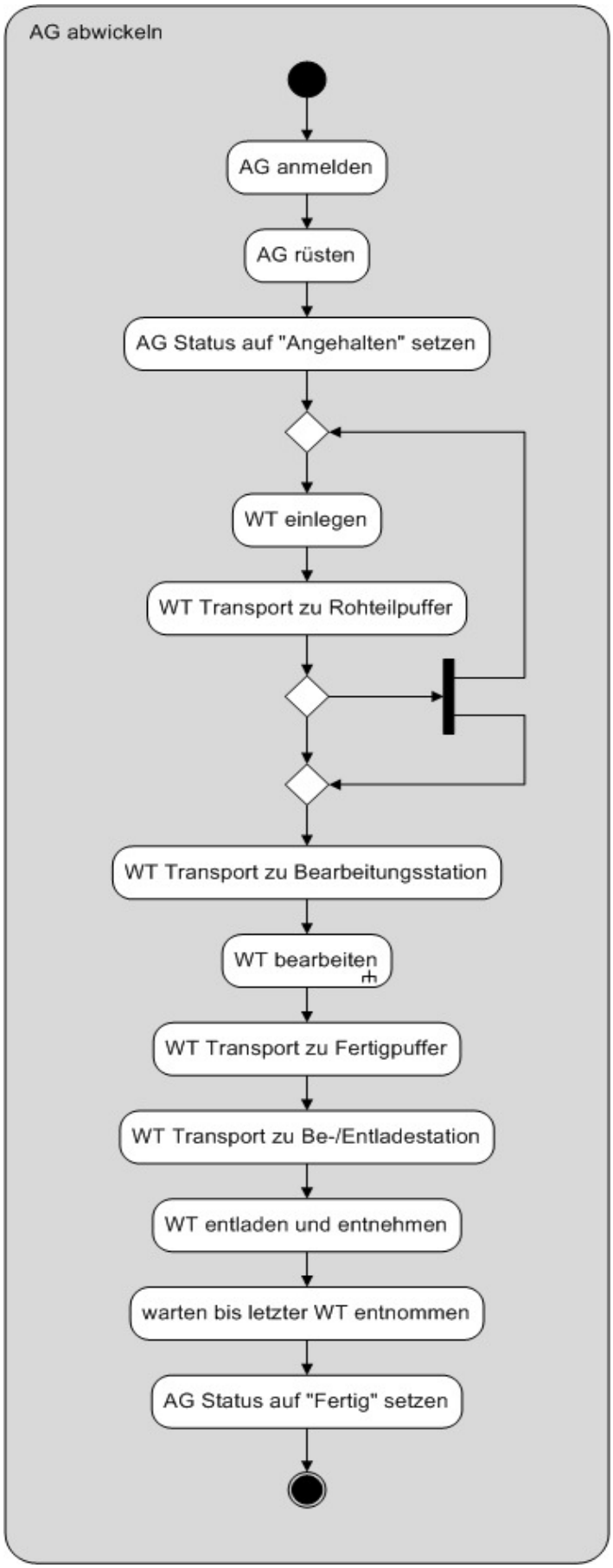

Abbildung 4.2: UML-Aktivitätendiagramm Arbeitsgang abwickeln

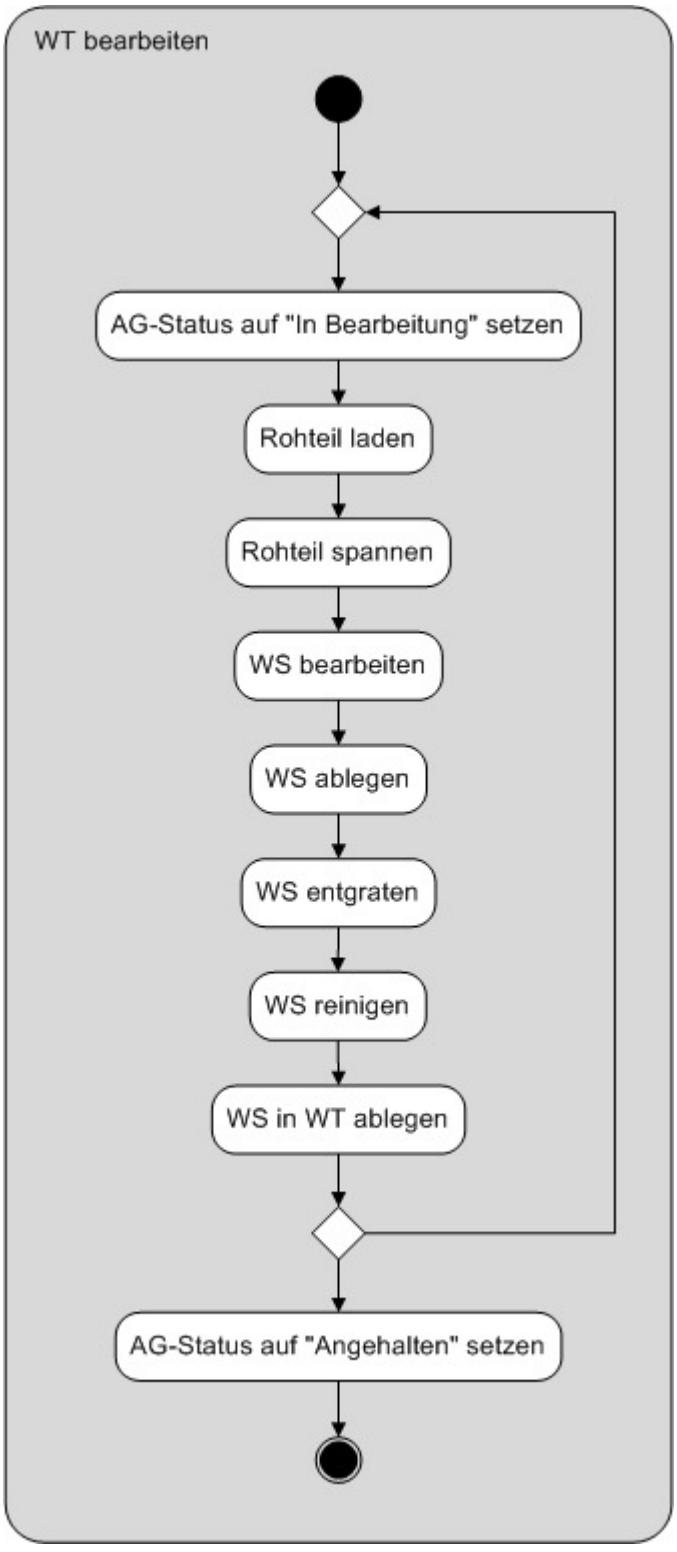

Abbildung 4.3: UML-Aktivitätendiagramm Werkstückträger bearbeiten

1. Auftrag freigeben, Auftrag übernehmen und Auftrag feinplanen

Vom ERP-System freigegebene Fertigungsaufträge werden an ein Feinplanungstool übergeben. Dieses Feinplanungstool plant den Auftrag dann terminlich genau ein. Weiters wird auch noch der erste Arbeitsgang des jeweiligen Fertigungsauftrages

freigegeben. Diese Schritte betreffen das Manufacturing Execution System nicht direkt, sind auch nicht Gegenstand dieser Diplomarbeit und deshalb auch nicht näher beschrieben.

2. Auftrag übergeben

Dies ist ebenfalls noch ein Schritt, der nicht vom MES ausgeführt wird. Es ist aber festzulegen, in welcher Form die Fertigungsaufträge übergeben werden. Dies geschieht in Form einer XML-Datei. Diese Datei muss ein genau definiertes Layout besitzen und wird in einen vordefinierten Ordner gespeichert. Wie das Layout auszusehen hat, ist in Punkt 8.1.3 detailliert beschrieben.

3. Auftrag ins MES übernehmen

Dieser Schritt betrifft zum ersten Mal die SIMATIC IT Production Suite und zwar die Komponente SIMATIC IT Data Integration Service. Dieses Service stellt Verbinder zur Verfügung, die ständig darauf warten, dass ein Fertigungsauftrag vom Feinplanungstool in den vordefinierten MES-Auftragsordner gespeichert wird.
Sobald also eine XML-Datei abgespeichert wird, wird diese abgerufen und der enthaltene Auftrag in den SIMATIC IT Production Order Manager übertragen. Die dazugehörige Logik wird im Production Modeler hinterlegt.

4. Arbeitsgang anmelden

Ist der Auftrag im Auftragsmanagement abgelegt, wird automatisch der jeweils freigegebene Arbeitsgang direkt an die zugehörige Produktionszelle bzw. Produktionseinheit weitergeleitet. Der Bediener dieser Zelle sieht an seinem Bedienterminal, dass ein neuer Arbeitsgang zu fertigen ist. Er kann nun den AG auswählen und anmelden. Beim diesem Vorgang wird der Arbeitsgangstatus im POM von „Freigegeben“ auf „Angemeldet“ gesetzt. „Angemeldet“ heißt, dass der Auftrag gerüstet werden kann bzw. ein Werkstückträger in die Zelle eingeschleust werden kann.

5. Arbeitsgang rüsten

Hier wird der Status des Arbeitsganges vom Bediener auf „Rüsten“ gesetzt. Dies kann nur geschehen, wenn kein Auftrag in Bearbeitung ist. Der Bediener kann nun die Anlage rüsten.

6. AG Status auf angehalten setzen

Ist der Arbeitsgang fertig gerüstet, wird der Status vom Bediener auf „Angehalten“ gesetzt, um später eine Zeitauswertung des Arbeitsganges einfacher zu gestalten.

7. Werkstückträger einlegen

Abhängig vom Produkt, das gefertigt werden soll, wird ein leerer Werkstückträger bestimmten Typs in die Be-/Entladestation eingelegt. Ist die Palette eingelegt, wird dies vom Bediener bestätigt.

8. WT-Transport zu Rohteilpuffer

Nachdem der Bediener das Einlegen eines Werkstückträgers in der Be-/Entladestation bestätigt hat, läuft die Palette über den Umsetzer in den Rohteilpuffer ein.

9. WT-Transport zu Bearbeitungsstation

Ist nun die Bearbeitungsstation leer, kann der WT in die Bearbeitungsstation eingeschleust werden. Die Palette ist nun bereit, mit WS beladen zu werden.

10. WT bearbeiten

Dieser Schritt umfasst mehrere Punkte, die teilweise vom MES und teilweise von der Anlagenautomatisierung übernommen werden. Zuerst wird der Status des Arbeitsganges auf „In Bearbeitung“ gesetzt. Anschließend sendet das MES ein Signal an die Anlage, dass ein Werkstück zu fertigen ist. Die folgenden Schritte bis inklusive „WS in WT ablegen“ werden dann von der Anlagenautomatisierung übernommen. Ist ein Werkstück abgelegt kontrolliert das MES, ob noch Teile zu fertigen sind. Ist dies der Fall, wird ein weiteres Signal an die Anlage geschickt. Ansonsten wird der Arbeitsgangstatus wieder auf „Angehalten“ gesetzt.

11. WT Transport zu Fertigpuffer

Wenn alle WS im WT abgelegt worden sind und der Status nicht mehr auf „In Bearbeitung“ ist, wird der WT für den Transport in den Fertigpuffer freigegeben. Ist kein WT im Puffer, wird die Palette aus der Bearbeitungsstation über den Umsetzer_2 in den Fertigpuffer geschleust.

12. WT Transport zu Be-/Entladestation

Der WT wird über das Förderband zur Be-/Entladestation transportiert, wenn diese nicht gerade aktiv ist.

13. WT entladen und entnehmen

In der Be-/Entladestation werden die fertigen WS und nötigenfalls auch der WT durch den Bediener entnommen. In weiterer Folge bestätigt der Bediener das durchgeführte Entladen.

14. Warten, bis alle WT entnommen

Ein Arbeitsgang kann aus mehreren Werkstückträgern bestehen, welche nacheinander bearbeitet werden. Um den Arbeitsgang auf Status „Fertig“ setzen zu können, müssen erst alle WT entladen und entnommen werden. Dies geschieht in diesem Schritt.

15. AG-Status auf Fertig setzen

Sind alle WS fertig, wird der AG-Status vom Bediener auf „Fertig“ gesetzt.

16. AG rückmelden

Ist ein Arbeitsgang fertig, wird dieser sofort an das ERP zurückgemeldet. Darum wird eine XML-Datei in einen vordefinierten Ordner kopiert, der vom ERP überwacht wird.

17. AG-Rückmeldung übernehmen

Das ERP übernimmt nun die Auftragsrückmeldung.

18. Nächsten AG freigeben

Gibt es einen Folgearbeitsgang, wird dieser vom MES automatisch freigegeben. Somit kann dieser Arbeitsgang gestartet werden.

19. Auftragsstatus auf Fertig setzen

Haben alle AG den Status „Fertig“, kann auch der ganze Auftrag den Status „Fertig“ erhalten.

20. Auftrag rückmelden

Das MES speichert nun eine XML-Datei in einen vorgegebenen Ordner. Diese Datei enthält die Rückmeldedaten für das ERP-System.

21. Auftragsrückmeldung übernehmen

Dieser Schritt betrifft nun wieder das ERP-System und nicht MES, deshalb ist er auch nicht mehr näher beschrieben.

5 SIMATIC IT Production Suite

Die bei dieser Diplomarbeit eingesetzte MES-Software heißt „SIMATIC IT Production Suite Version 6“ und ist ein Produkt der Firma Siemens. Sie ist eine hoch integrierte Plattform von Komponenten und erhebt den Anspruch ISA-95-komform zu sein.

5.1 Aufbau und Schnittstellen des Systems

Das Kernstück der SIMATIC IT Production Suite ist der Production Modeler. Die Software beinhaltet neben dem Production Modeler auch noch eine Anzahl von Komponenten. Jeder Bestandteil bietet eine Funktionalität, die einem oder mehreren ISA-95 Elementen entspricht. Jede Komponente besteht aus einem Server, der Methoden der jeweiligen Komponente zur Verfügung stellt, sowie einer dazugehörigen Bedieneroberfläche. In Abbildung 5.1 wird die Architektur der SIMATIC IT Production Suite dargestellt.

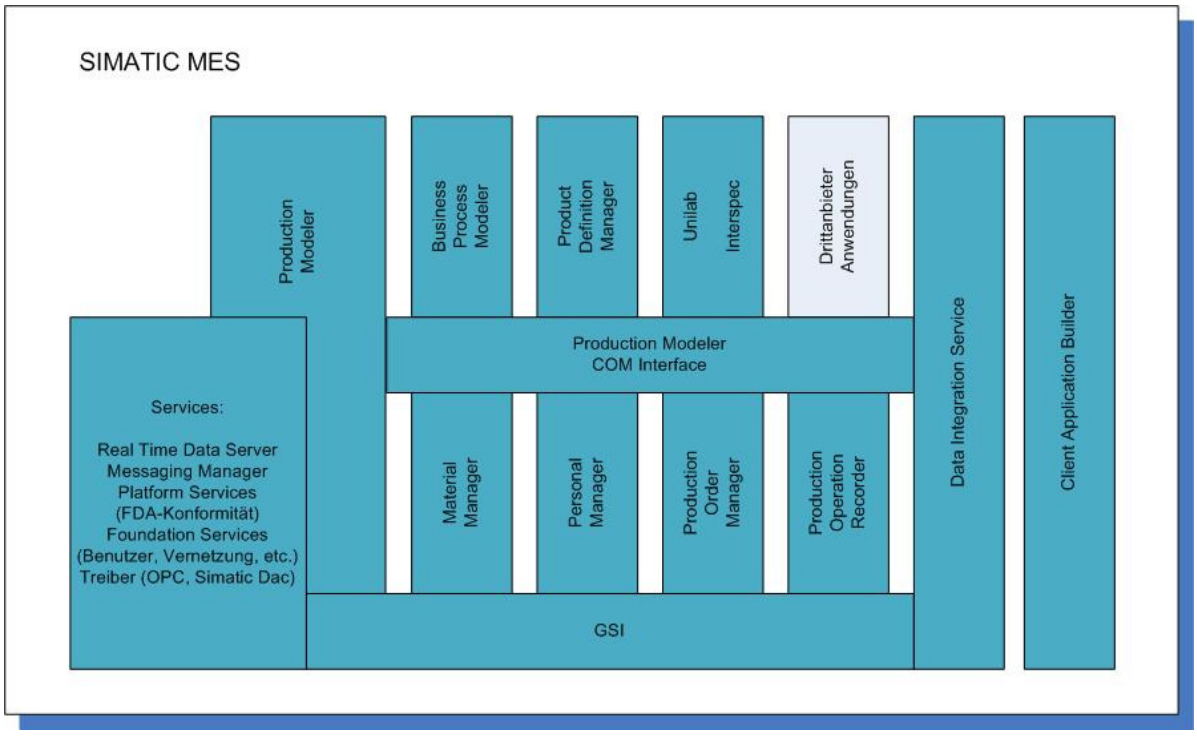

Abbildung 5.1: Architektur von SIMATIC IT Production Suite[5]

[5] Quelle: Siemens AG (2005).

SIMATIC IT Komponenten können mit dem Production Modeler über folgende Schnittstellen kommunizieren:

- Production Modeler COM Interface
- GSI, Global Settings Interface

Alle Bestandteile des Programms werden im folgenden Abschnitt genauer erklärt.

5.2 Production Modeler

Der SIMATIC IT Production Modeler (PM) ist der wichtigste Bestandteil der SIMATIC IT Production Suite, die dem Benutzer das Modellieren des Unternehmens und seiner Prozesse ermöglicht. Im PM werden die Anlage und deren Prozesse aber nicht nur modelliert, sondern er dient auch als Prozess Engine. Die Prozesse werden hier in Form von Regeln abgebildet. Diese Regeln rufen über eine der beiden Schnittstellen (COM oder GSI) Methoden auf, die von den Komponenten zur Verfügung gestellt werden, siehe Abbildung 5.2.

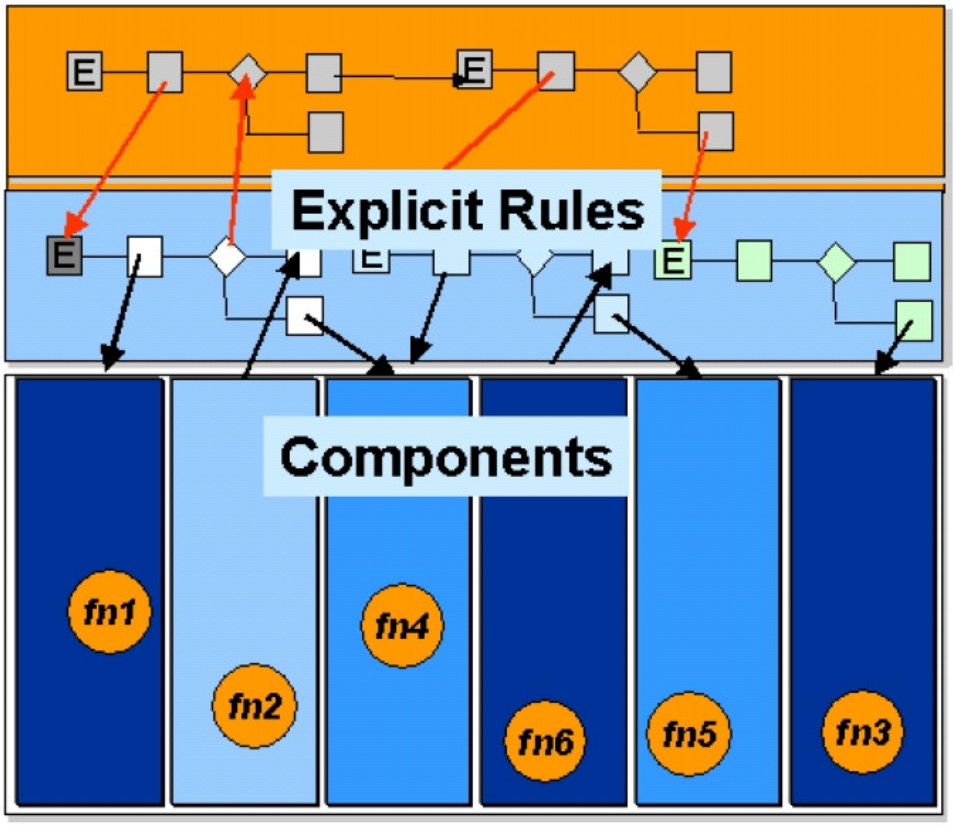

Abbildung 5.2: Kommunikation zwischen PM und den Komponenten[6]

[6] Quelle: Mezzaro, F./Ferrari, R./Pazzini, M.: (2005:8).

5.2.1 Hauptbereiche des Production Modelers

Der PM hat drei Hauptbereiche, den Bibliotheksbereich, den Arbeitsbereich und den Ausführungsbereich.

5.2.1.1 Bibliotheksbereich

Im Bibliotheksbereich wird die Leistungsfähigkeit jeder Maschine und Applikation in der zu modellierenden Anlage erfasst und abgebildet. Dies geschieht hierarchisch, Ebene für Ebene, beginnend bei den einzelnen Units, über die verschiedenen Cells bis hin zu den Areas und Sites. In weiterer Folge werden Regeln erstellt, welche die Prozesse abbilden. Alle Elemente, die man hier modelliert, können in einer eigenen Bibliothek abgespeichert und dann im Arbeitsbereich als Komponenten der Anlage (Plant) eingefügt werden. Elemente, die in der Bibliothek erstellt worden sind, können beliebig oft eingesetzt werden. Gibt es in einem Unternehmen mehrere gleiche Fertigungsanlagen, müssen sie also nur einmal modelliert werden.

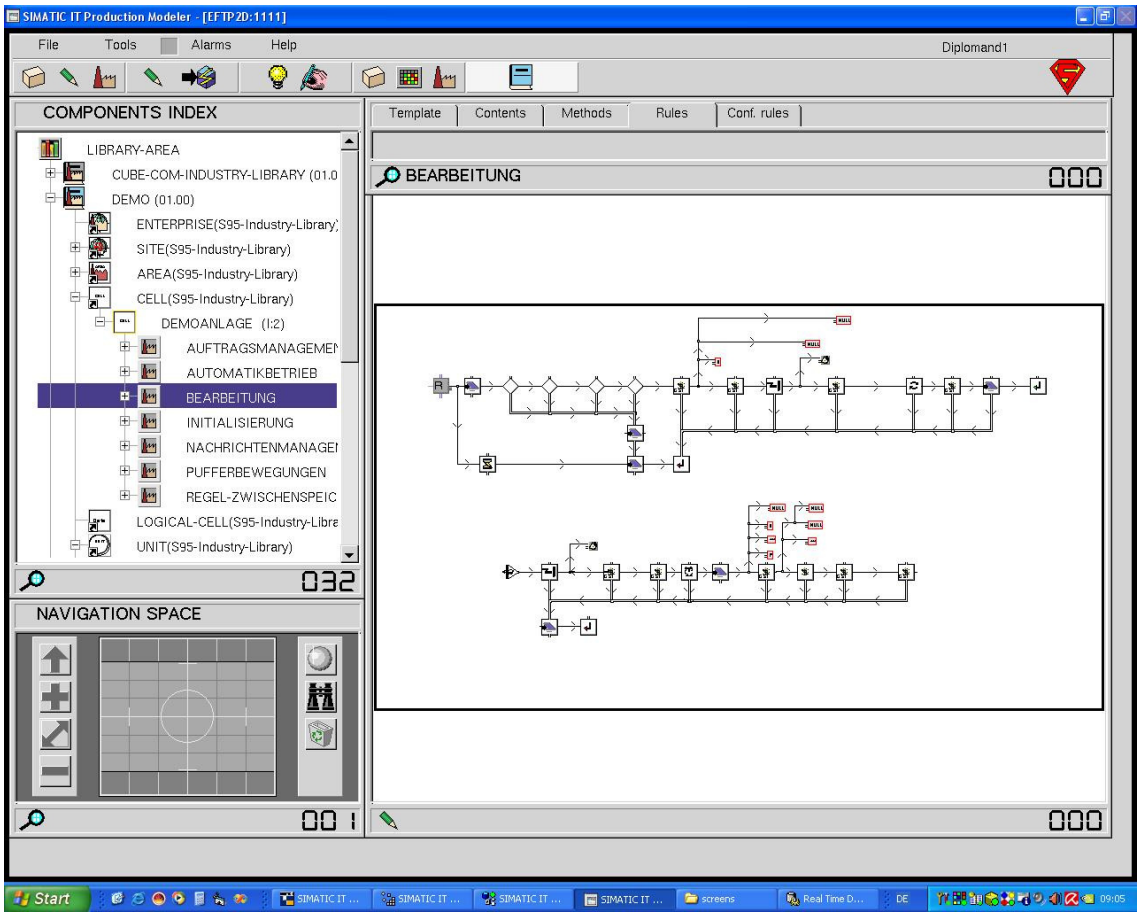

Abbildung 5.3: Bibliotheksbereich im PM

5.2.1.2 Arbeitsbereich

Im Arbeitsbereich wird dann die tatsächliche Anlage abgebildet. Hier werden die entwickelten Bibliotheken verwendet und Elemente daraus eingefügt und instanziert. Auch die Prozesse laufen im Arbeitsbereich ab. Weiters wird er verwendet um Regeln zu testen, die im Bibliotheksbereich erstellt wurden und in der Anlage durch das Einfügen eines Elements integriert sind. Um eine Regel zu testen, muss sie im Arbeitsbereich aktiviert werden.

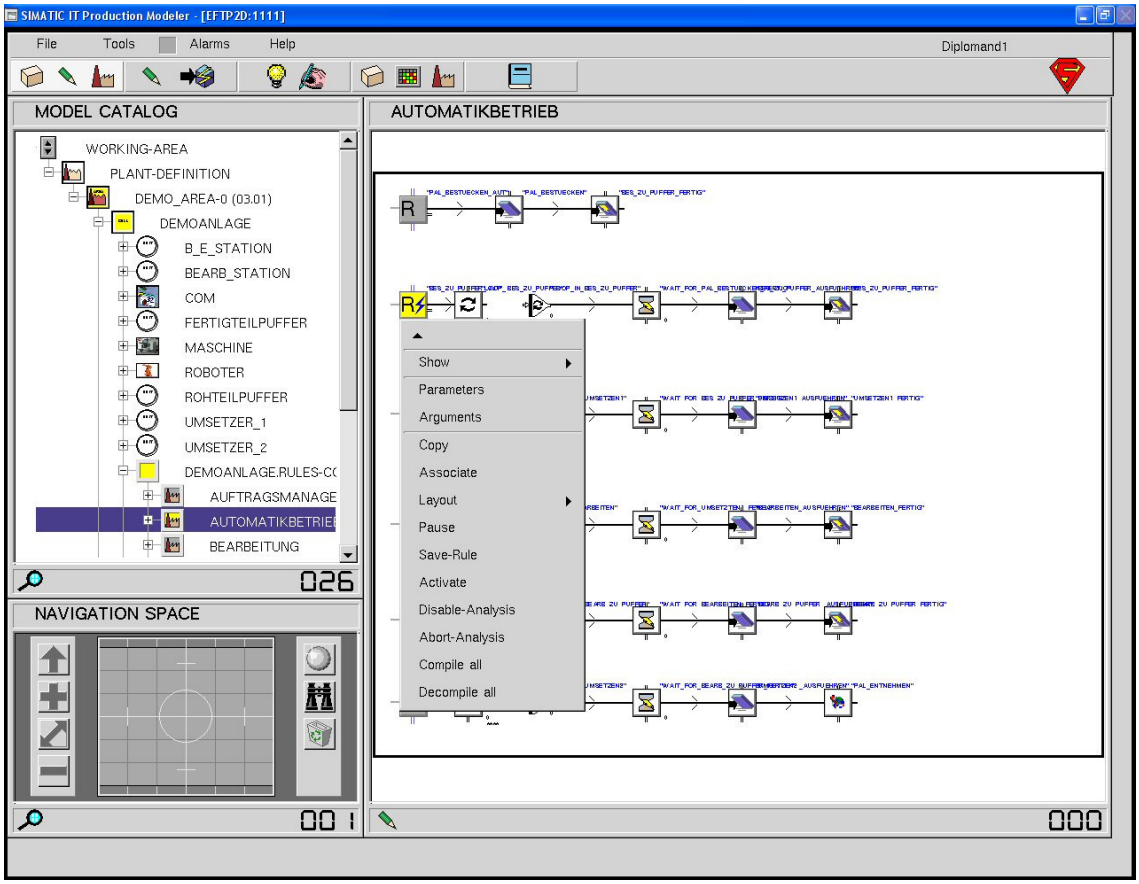

Abbildung 5.4: Arbeitsbereich im PM

5.2.1.3 Ausführungsbereich

Der Ausführungsbereich dient als Analysetool während der Testphase der abgebildeten Prozesse. Man sieht hier die Regeln, welche im Arbeitsbereich aktiviert, d.h. gestartet wurden. Die farbliche Kennung der Objekte in einer Regel lässt Schlüsse über den konkreten Zustand zu. Es gibt folgende Möglichkeiten der farblichen Darstellung:

- weiß: Objekt ist noch nicht gestartet worden, da die Regel vorher irgendwo angehalten oder abgebrochen wurde
- gelb: gerade aktiv, aber noch nicht vollständig ausgeführt,
- grün: Objekt wurde bereits ausgeführt und es gab keinen Fehler, d.h. korrekte Durchführung,

- rot: Objekt konnte nicht fehlerfrei ausgeführt werden, die Regel wird beim negativen Ausgang des Objekts fortgesetzt.

In Abbildung 5.5 sieht man eine Regel, die gerade aktiv ist, aber noch nicht vollständig durchlaufen wurde.

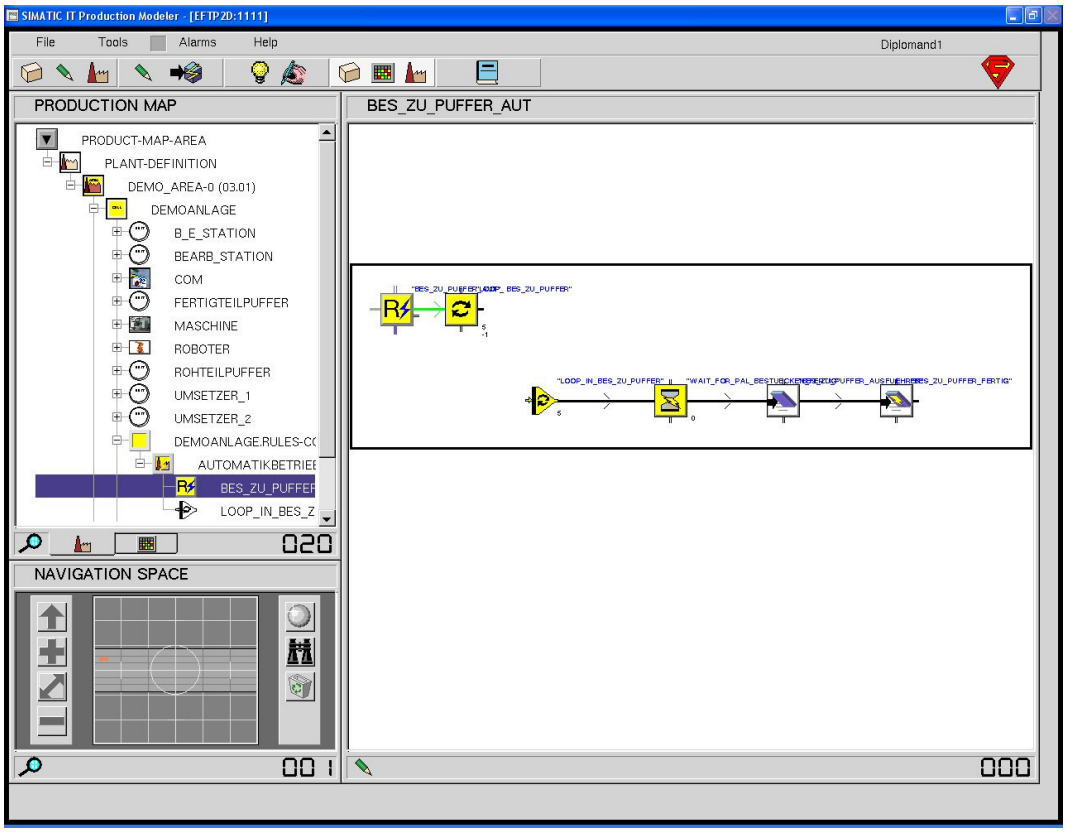

Abbildung 5.5: Ausführungsbereich im PM

5.3 Real Time Data Engine

In Abbildung 5.6 sieht man die Architektur von der Real Time Data Engine.[7]

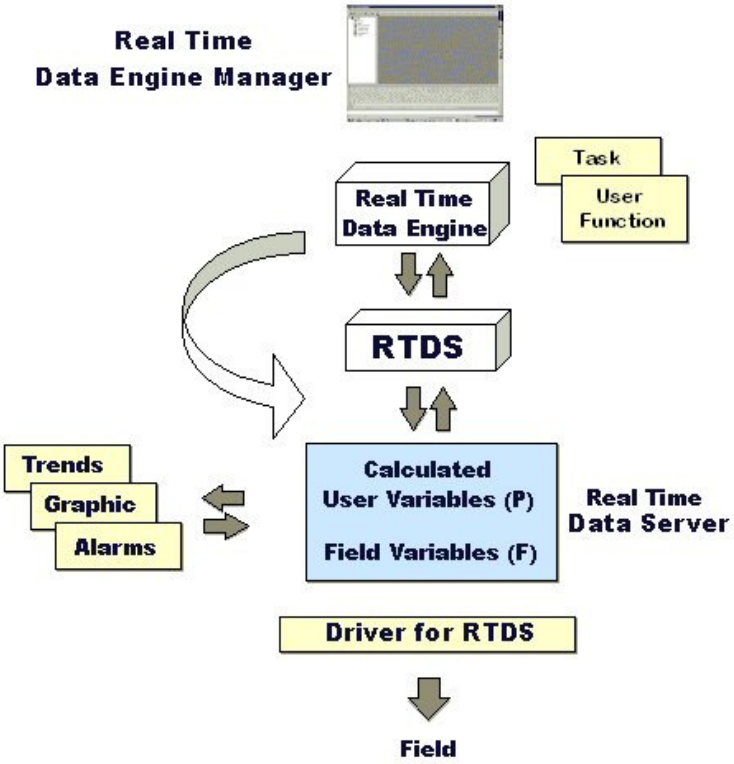

Abbildung 5.6: Architektur der Real Time Data Engine

Mit der Real Time Data Engine kann man benutzerdefinierte Variablen, so genannte Cube-Variablen, konfigurieren und bearbeiten. Erstellt werden sie jedoch im Production Modeler als Attribute von einer Unit oder Cell. Die Daten können hier von verschiedenen Geräten (z.B. SPS) mit Hilfe passender Schnittstellen eingelesen werden. Weiters können auch Daten auf diese Geräte geschrieben werden. Die Kommunikation funktioniert also in beide Richtungen. So kann man zum Beispiel Aus- und Eingänge einer Steuerung mit der MES-Software verbinden. Die Konfiguration und Bereitstellung der Echtzeitdaten für die Komponenten von SIMATIC IT Production Suite erfolgt im Real Time Data Engine Manager, der die Bedieneroberfläche darstellt (Abbildung 5.7).

[7] Quelle: Siemens Hilfefiles zu SIMATIC IT Real Time Data Engine Manager.

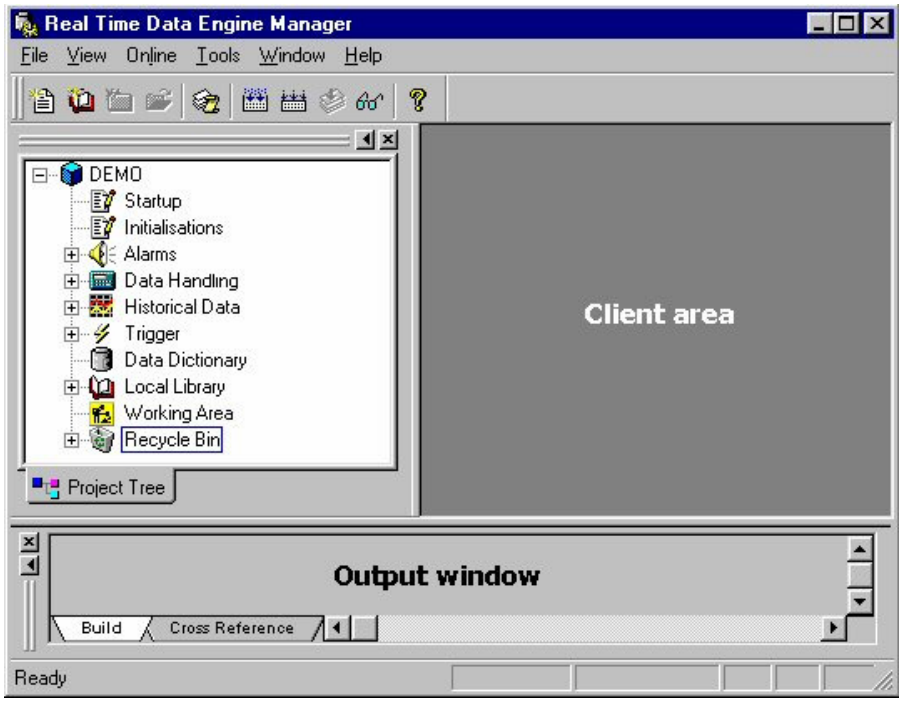

Abbildung 5.7: Real Time Data Engine Manager

5.4 Order Management

Das Auftragsmanagement ist eine sehr wichtige Komponente eines MES und wird in der SIMATIC IT Production Suite vom Production Order Manager (POM) übernommen. Folgende Arbeiten können der Bedieneroberfläche des POM durchgeführt werden:

- Importieren von Aufträgen aus einem ERP System mit Hilfe von XML- oder TXT-Dateien
- Statusverwaltung
- Automatische Weiterleitung von Aufträgen
- Überwachung und Aktivierung durchgeführter Aufträge

Im Production Order Manager werden Aufträge aus dem Scheduler übernommen und durch den Production Modeler an die Fertigung übermittelt.

5.5 Material Management

Der SIMATIC IT Material Manager (MM) ist für die Materialverwaltung verantwortlich. Nach dem ISA 95 Standard kann man Materialien definieren und hierarchisch ordnen. Es kann jederzeit nachvollzogen werden, welches Material in welcher Menge vorhanden ist und wo sich jedes Los gerade befindet. Die Daten hierfür werden in einer SQL Server 2000 Datenbank hinterlegt. Man kann neben Losen und

Unterlosen auch so genannte Handling Units (HUTs) definieren, in denen mehrere Lose zusammengefasst und verwaltet werden können.

5.6 Messaging Manager

Zur Verwaltung von Daten- und Nachrichtenanforderungen innerhalb des SIMATIC IT Systems beinhaltet die Production Suite den Messaging Manager. Er besteht im Wesentlichen aus einem Server zur Verwaltung der Nachrichten.

Bei dieser Diplomarbeit wird der Messaging Manager dazu verwendet, um aus dem Production Modeler Meldungsfenster zu generieren. Mit Hilfe dieser Fenster können Informationen am Bildschirm des Benutzers angezeigt und auch benötigte Eingabeaufforderungen realisiert werden. Das Layout und die Übergabewerte dieser benutzerdefinierten Meldungsfenster können mit Hilfe eines Data Template Editors gestaltet werden.

5.7 DIS

Das Tool Data Integration Service (DIS) der SIMATIC IT Produciton Suite ermöglicht den Nachrichtenaustausch von Anwendungen unterschiedlichen Typs. Es stellt also eine allgemeine Datentransportschicht dar. Nachrichten, die das DIS verwendet, sind XML-Textnachrichten.

Eine der Schlüsselaufgaben des Data Integration Service ist der Datenaustausch des MES mit einem ERP-System. Dies geschieht also mit Hilfe von XML-Dateien. In dieser Diplomarbeit wird das DIS dazu verwendet, um Aufträge aus einem ERP-System in den SIMATIC IT Production Order Manager einzutragen. Näheres zum Aussehen (Struktur) einer XML-Datei ist in Kapitel 8.1.3 zu finden.

5.8 Client Application Builder

Um eine geeignete Oberfläche für den Bediener der Fertigungszelle zu programmieren stellt SIMATIC IT Production Suite ein Plug-In (CAB) für die Programmiersoftware Microsoft Visual Studios.NET 2003 zur Verfügung.

5.9 Weitere Komponenten

SIMATIC IT Production Suite beinhaltet noch weitere Komponenten, welche Standardfunktionen eines vollständigen Manufacturing Execution Systems abbilden. In dieser Diplomarbeit finden folgende Komponenten jedoch keine Verwendung:

- Personal Manager, dient zur Verwaltung von Personaldaten und Arbeitszeiten, sowie die Verwaltung von Anwendergruppen.
- Report Manager, ein Tool, um Datenreports aus der SQL Datenbank zu erstellen.

- Production Definition Manager, ein Engineering Modul um zu definieren, wie ein Produkt hergestellt wird. Diese Komponente ist in erster Linie für verfahrenstechnische Produktionsabläufe gedacht, deshalb ist ihr Einsatz in der diskreten Fertigung nicht sinnvoll.

6 Erste Schritte in der SIMATIC IT Production Suite

In der SIMATIC IT Production Suite gibt es ein Management-Tool, von dem aus die einzelnen Bestandteile des Programms gestartet werden können. Es gibt hier auch die Möglichkeit, Voreinstellungen für den Start des Programms zu treffen. Dieses Tool wird „Management Console“ genannt. Wie man damit umgeht und welche Einstellungen zu treffen sind, wird in diesem Abschnitt beschrieben.

6.1 Bedienung der SIMATIC IT Management Console

Die Managementkonsole ist das eigentliche Hauptfenster der SIMATIC IT Production Suite (Version 6.1 Service Pack 1), doch um zu dieser Bedieneroberfläche zu gelangen müssen zuvor noch ein paar Schritte durchgeführt werden.

6.1.1 Starten von SIMATIC IT

Nach dem Start der SIMATIC IT Services muss man sich als User anmelden, es wird jedoch kein Fenster automatisch geöffnet, in dem man dazu aufgefordert wird. Man muss die Tastenkombination Shift-Esc drücken, um das „User Logon“ Fenster zu öffnen, welches in Abbildung 6.1 dargestellt ist.

Abbildung 6.1: User Logon Fenster

Nach erfolgreicher Anmeldung wird das Fenster „Plant Management“ geöffnet, siehe Abbildung 6.2. Die Karteikarte „Plant Tree“ wird angezeigt (links unten), auf der nach vorhandenen Anlagen (Plants) gesucht werden kann. Durch Klicken der rechten Maustaste auf die gewünschte Anlage, öffnet sich ein Pop Up Fenster, wo man den Befehl „Logon to Management Console“ auswählen kann. Selbiges erreicht man durch Auswahl einer Anlage, welche dann blau hinterlegt ist, und Klicken auf die Menüleiste File – Logon to Management Console.

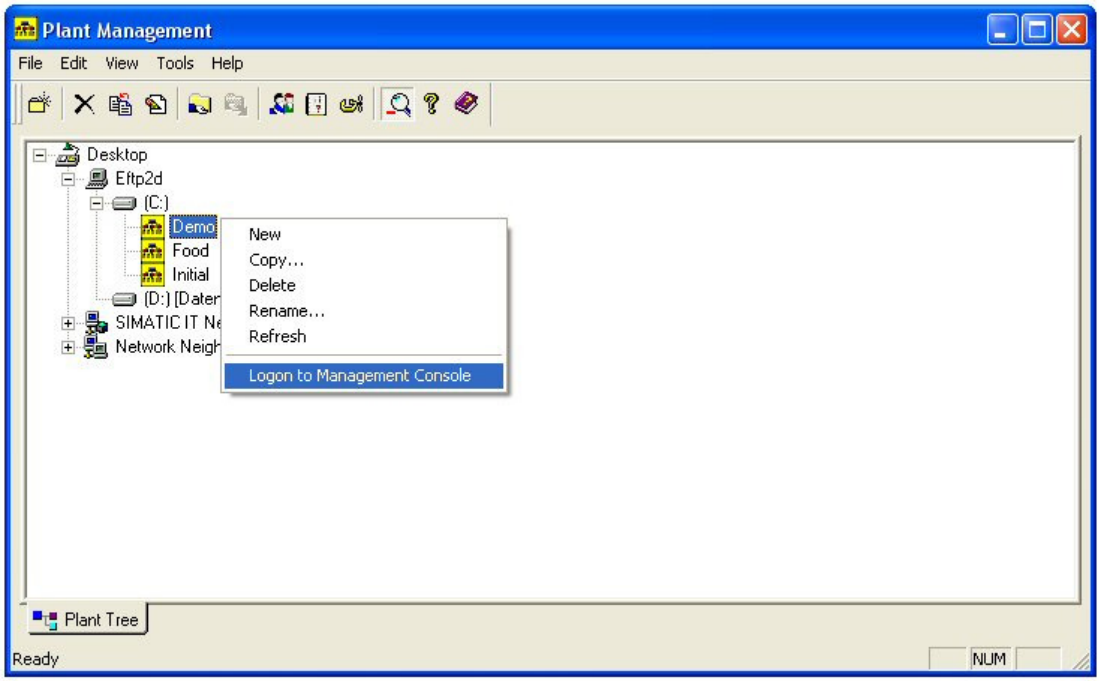

Abbildung 6.2: Plant Management Fenster

In diesem Projekt heißt die Anlage „DEMO“. In der Managementkonsole können dann die benötigten Komponenten gestartet werden. Dies geschieht entweder über die Menüleiste oder die Karteikarteikarte „Components“, siehe Abbildung 6.3.

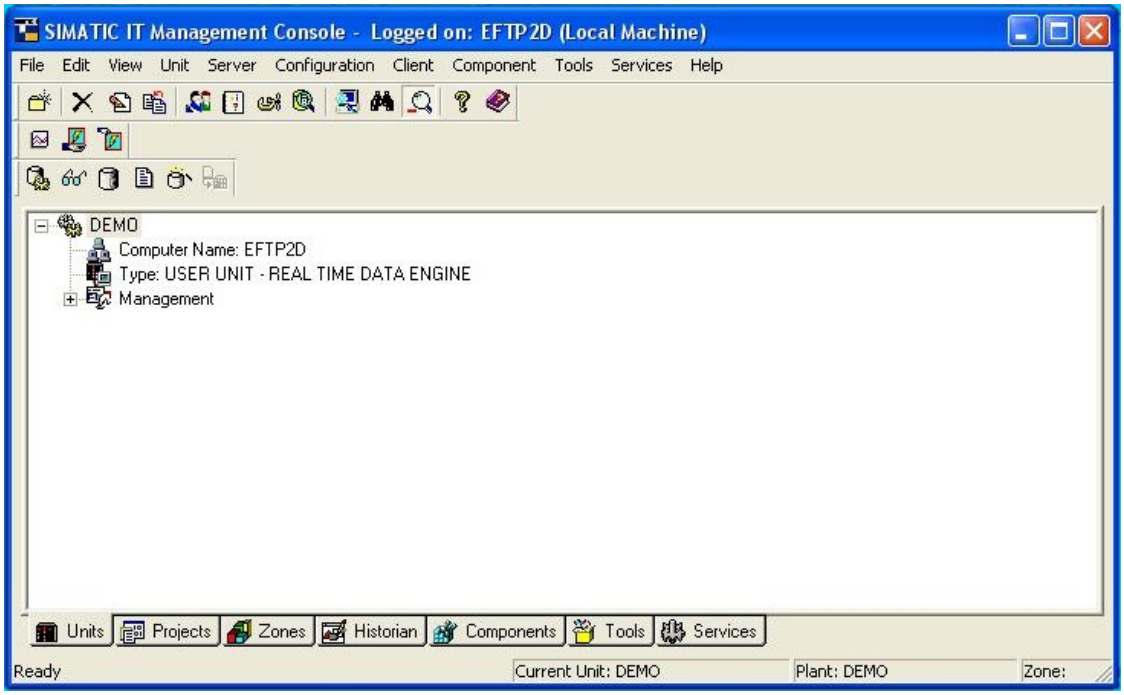

Abbildung 6.3: SIMATIC IT Management Console

6.1.2 Voreinstellungen definieren

Viele Komponenten beziehungsweise deren Server können auch automatisch gestartet werden. Diese Einstellungen sind in der Menüleiste unter Configuration – System zu treffen und durch Anklicken öffnet sich ein Fenster mit dem Namen „System Configuration“. In der Karteikarte „Start-Up“ kann der User festlegen, welche Applikationen von SIMATIC IT automatisch gestartet werden sollen. Durch Anklicken der Check Box „Automatic Log–On to Current Plant“ erspart man sich die zuvor beschriebe Anmeldeprozedur bei einem neuerlichen Start von SIMATIC IT Services, siehe Abbildung 6.4.

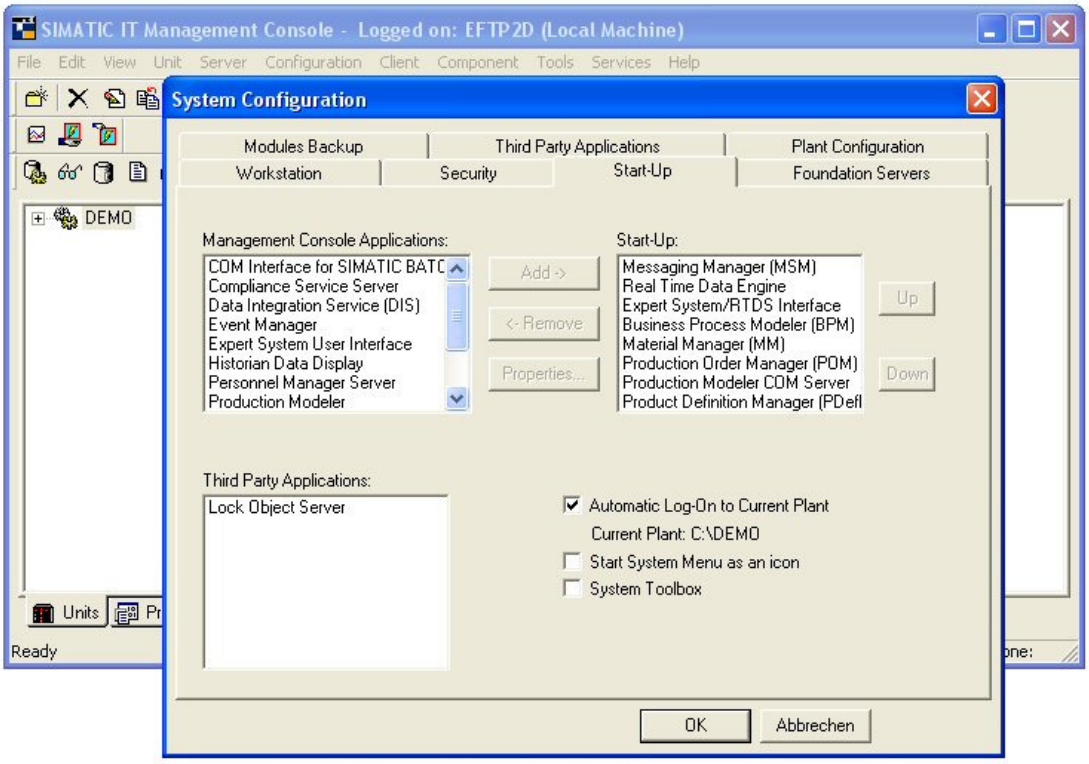

Abbildung 6.4: System Configuration

6.1.3 Starten des Production Modelers und der Komponenten

Man kann den Production Modeler und alle anderen SIMATIC IT Komponenten direkt aus der Managementkonsole starten. Dazu geht man in der Menüleiste auf „Components“ und wählt die entsprechende Komponente aus. Bevor man die Benutzeroberfläche einer Komponente starten kann muss auch der jeweilige Server gestartet sein. In Abbildung 6.5 ist als Beispiel der Start des PM zu sehen.

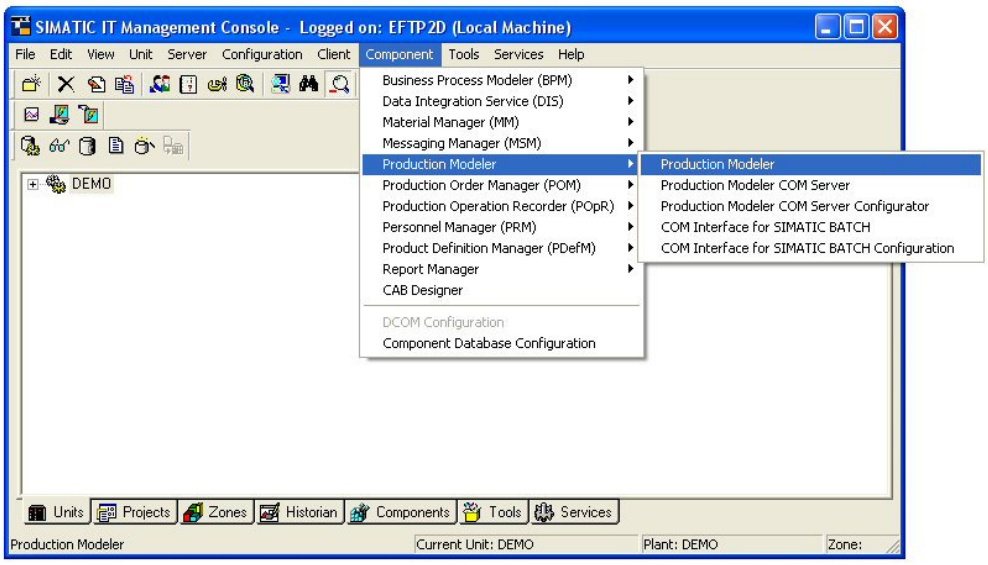

Abbildung 6.5: Starten des Production Modelers

Anschließend öffnet sich das entsprechende Bearbeitungsfenster der ausgewählten Komponente. Die Managementkonsole bleibt im Hintergrund immer aktiv. Es können parallel auch mehrere Komponenten laufen.

7 Bedienung des Production Modelers und Modellieren der Anlage

7.1 Einleitung

Im Production Modeler wird ein Modell der Anlage erstellt. Hierzu wird zuerst im Bibliotheksbereich eine Bibliothek mit allen Anlagenelementen (Units, Cells, …) erstellt und gespeichert. Falls mehrere gleiche Anlagenkomponenten im Modell vorkommen, müssen diese nur einmal erstellt werden und können dann beliebig oft verwendet werden. Die Elemente sollen die gleichen Eigenschaften (Attribute, Methoden und Events) haben wie die der physischen Anlage. Im Bibliotheksbereich werden in weiterer Folge noch Regeln erstellt, welche die Prozesse, die in der Anlage ablaufen, darstellen.

Wenn alle Anlagenelemente und die zugehörigen Regeln im Bibliotheksbereich modelliert sind, kann noch nicht damit gearbeitet werden. Es muss zuerst die Anlage im Arbeitsbereich erstellt werden. Über diesen Bereich werden dann alle Prozesse der Anlage gesteuert. Der Production Modeler fungiert also nicht nur als Modellierungstool sondern ist auch für die Prozessabläufe verantwortlich. Er dient also auch als Prozess-Engine.

Im folgenden Teil wird beschrieben, wie der Production Modeler zu bedienen ist und wie die genaue Vorgehensweise beim Modellieren und Erstellen der Anlage im PM aussieht.

7.2 Start des Production Modelers

Der Production Modeler wird über die SIMATIC IT Management Console gestartet. Während des Startvorganges wird das Fenster „Production Modeler Launcher“ angezeigt. Hier kann man eine „User KB“ auswählen, die gestartet werden soll, siehe Abbildung 7.1. „User KB“ ist eine Datei, in der alle Informationen zu einem Projekt gespeichert sind. Hat man noch kein eigenes Projekt angelegt, wählt man die „MESAD-ROOT.kb“ aus und speichert diese gleich unter einem neuen Namen wieder ab. Die „MESAD-ROOT.kb“ beinhaltet alle Basic Libraries des Production Modelers.

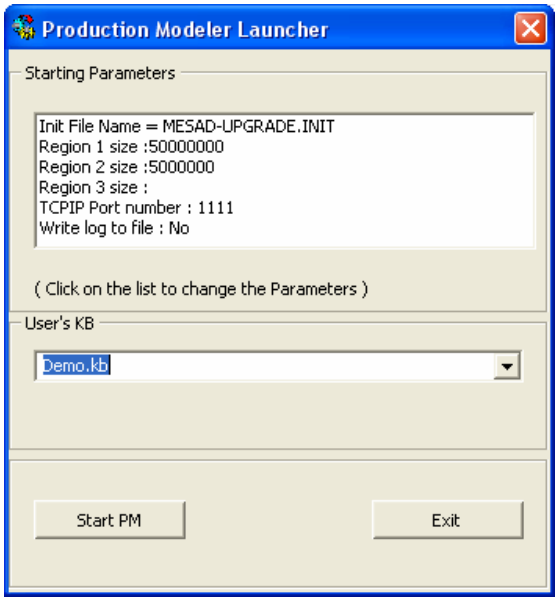

Abbildung 7.1: PM Launcher

Dieses Projekt heißt „Demo.kb“ und wird wie in Abbildung 7.1 ausgewählt. Durch Klicken auf den Button „Start PM“ wird die „DEMO.kb“ im Production Modeler gestartet. Sie ist ebenfalls aus der „MESAD-ROOT.kb“ erstellt worden.

7.3 Modellieren der Anlage im Bibliotheksbereich

Eine Anlage ist hierarchisch in Units, Cells, Areas und Sites eingeteilt, wobei die Hierarchie ganz unten mit Units beginnt. Die nächst höhere Ebene ist eine Cell, die mehrere Units enthalten kann. Danach folgt die Area, die wiederum aus mehreren Cells und Units bestehen kann. Ganz oben steht dann die Site, in der mehrere Areas enthalten sein können. In diesem Projekt sieht die Struktur wie im oberen Teil von Abbildung 7.2 aus. In der Abbildung ist auch die Zuordnung der einzelnen Units zur realen Anlage ersichtlich. Die jeweils zusammengehörigen Elemente sind mit gleichen Buchstaben gekennzeichnet.

Weiters kann man erkennen, dass manche Units (Unit-1, Unit-3, Unit-6) mehrmals vorkommen. Wenn man mehrere Objekte des gleichen Typs hat, werden diese also nur einmal erstellt und können dann beliebig oft in die Zelle geladen werden. Jedes einzelne Element muss aber eine genaue Bezeichnung besitzen, um Verwechslungen vorzubeugen.

Die Unit-2 („COM“) ist kein physischer Bestandteil der Fertigungszelle. Diese Einheit stellt die Verbindung zum COM-Wrapper her, welcher die Methoden zur Kommunikation mit der Anlage beinhaltet.

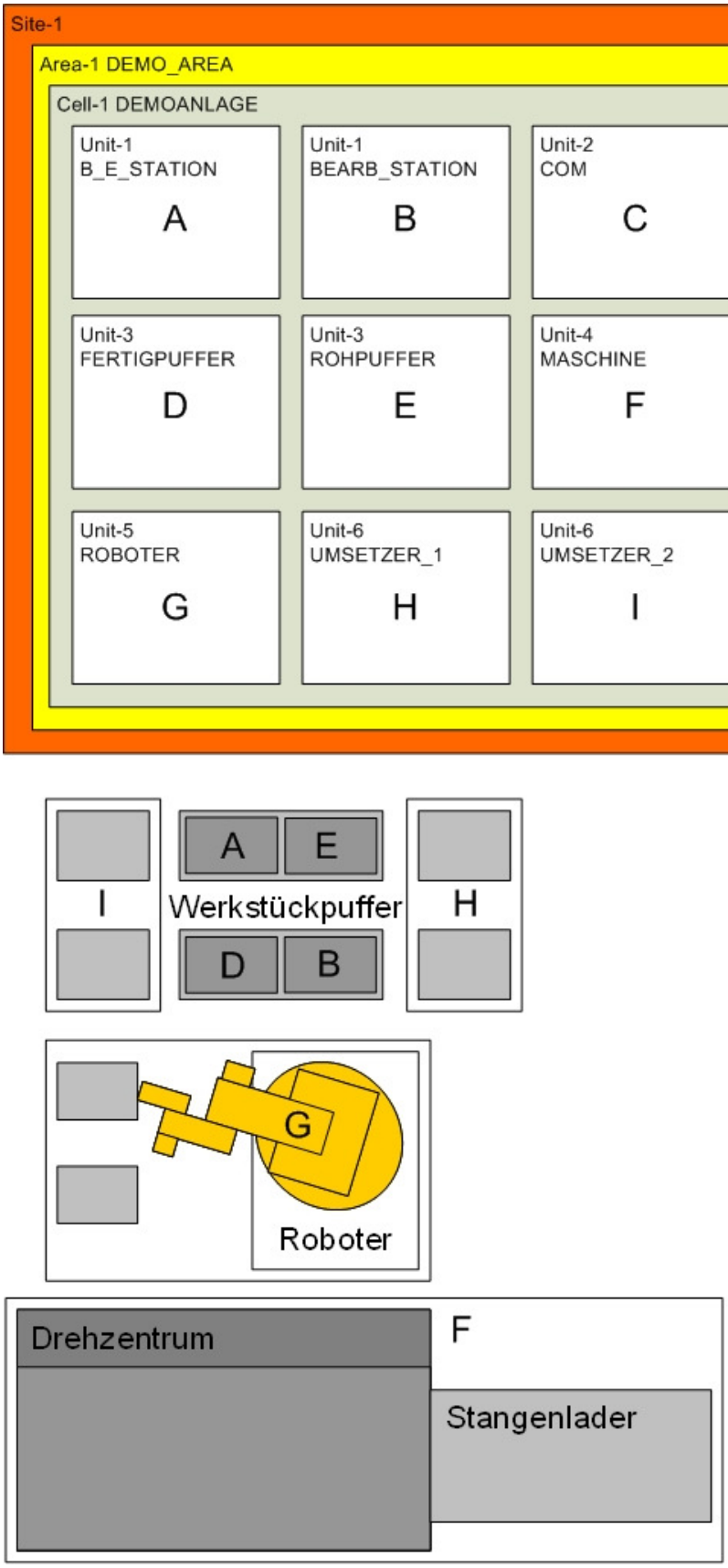

Abbildung 7.2: Klassenstruktur der Anlage

7.3.1 Erstellen einer neuen Bibliothek

Bibliotheken enthalten Komponentengruppen, die für die jeweilige Fertigung relevant sind. In einer solchen Bibliothek müssen also alle benötigten Objekte modelliert werden. Eine Benutzerbibliothek muss aber erst erstellt werden. SIMATIC IT bietet auch vorbereitete Bibliotheken an, in denen beispielsweise Standardmaschinen enthalten sind. In dieser Arbeit soll aber die ganze Anlage von Grund auf modelliert werden.

Um also eine eigene Bibliothek zu erstellen, klickt man mit der rechten Maustaste auf den Stamm der Baumstruktur im Bibliotheksbereich (Abbildung 7.3) und wählt dort „New“ aus. Im folgenden Fenster gibt man dann nur einen Namen für die Bibliothek ein und klickt auf „OK“. In diesem Projekt ist der Name der Bibliothek „DEMO“.

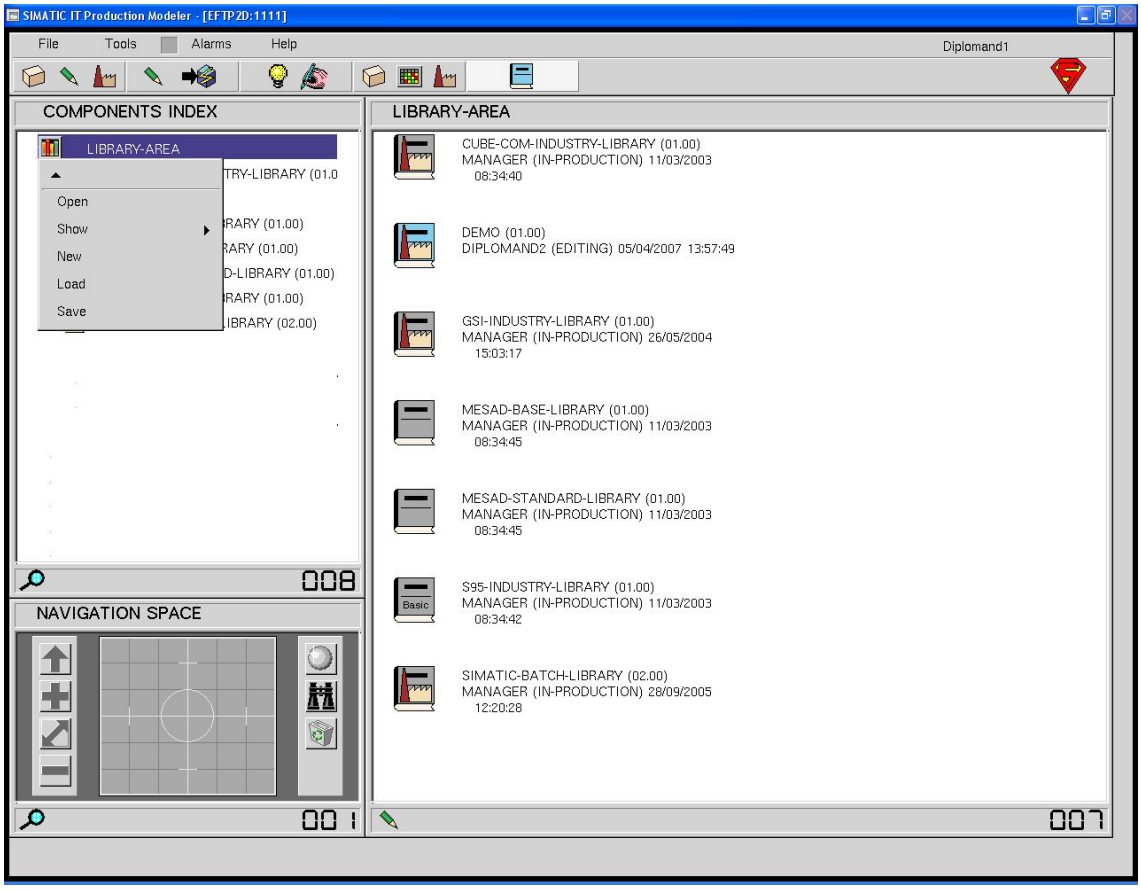

Abbildung 7.3: Stamm der Bibliotheksbaumstruktur

Ist die Bibliothek erstellt, wird mit dem Modellieren der einzelnen Anlagenobjekte begonnen. Man startet dabei hierarchisch gesehen ganz unten, also mit den Units.

7.3.2 Erstellen der Units

Units sind, wie zuvor erwähnt, in der PM-Hierarchie die untersten Elemente und Bestandteile der späteren Zellen. Aus diesem Grund müssen sie zuerst modelliert werden.

Anhand der Unit „B_E_STATION“ wird die Erstellung eines solchen Elements genau beschrieben. In weiterer Folge wird noch die „COM“-Unit genau beschrieben, da sie keine herkömmliche Unit der Anlage ist. Die anderen Units sind nur kurz erklärt.

- Unit-1 (B_E_STATION und BEARBSTATION)

  Diese Unit wird in der Anlage als B_E_STATION und BEARBSTATION verwendet. Dies sind zwei gleiche Elemente in der Anlage. Einerseits bilden sie die Be-/Entladestation zum Beschicken der Anlage mit Leerpaletten und zum entnehmen fertiger Paletten, andererseits die Bearbeitungsstation, wo die gefertigten Werkstücke auf die Palette gelegt werden.

  Diese Einheit kann folgende Eigenschaften besitzen:

  - Aktiv: Die Be/-Entladestation ist aktiv, wenn das Förderband der Station gerade in Betrieb ist.
  - Zustand: Der Zustand legt fest, ob sich die Hebestation gerade in der oberen oder unteren Endposition befindet.
  - Beladen: Beladen legt fest, ob eine Palette in der Station liegt.
  - Paletten_ID: Dieses Attribut beschreibt die Identität der Palette.

  Jetzt beginnt man mit dem eigentlichen Modellieren der Unit. Dazu navigiert man in der Baumstruktur im Bibliotheksbereich des Production Modelers zum Eintrag „Unit“. Hier klickt man mit der rechten Maustaste und wählt „Create-Subclass“ aus. Es öffnet sich ein Fenster in das man den „Class-Name“ eingeben muss. In diesem Fall ist das „B_E_STATION“. Die neue Unit wurde somit erstellt.

  Als nächster Schritt sind noch die Attribute der Unit festzulegen. Diese sind Eigenschaften der jeweiligen Einheit. Man navigiert also in der Baumstruktur zu der gewünschten Unit, klickt diese mit der rechten Maustaste und wählt dann Modify-Class – Modify-Attributes aus. Es öffnet sich das Fenster „Browser“ (Abbildung 7.4). Hier sieht man vordefinierte und selbst erstellte Attribute. Um ein benutzerdefiniertes Attribut hinzuzufügen, klickt man auf „Add“ und das Parameterfenster in Abbildung 7.4 wird angezeigt.

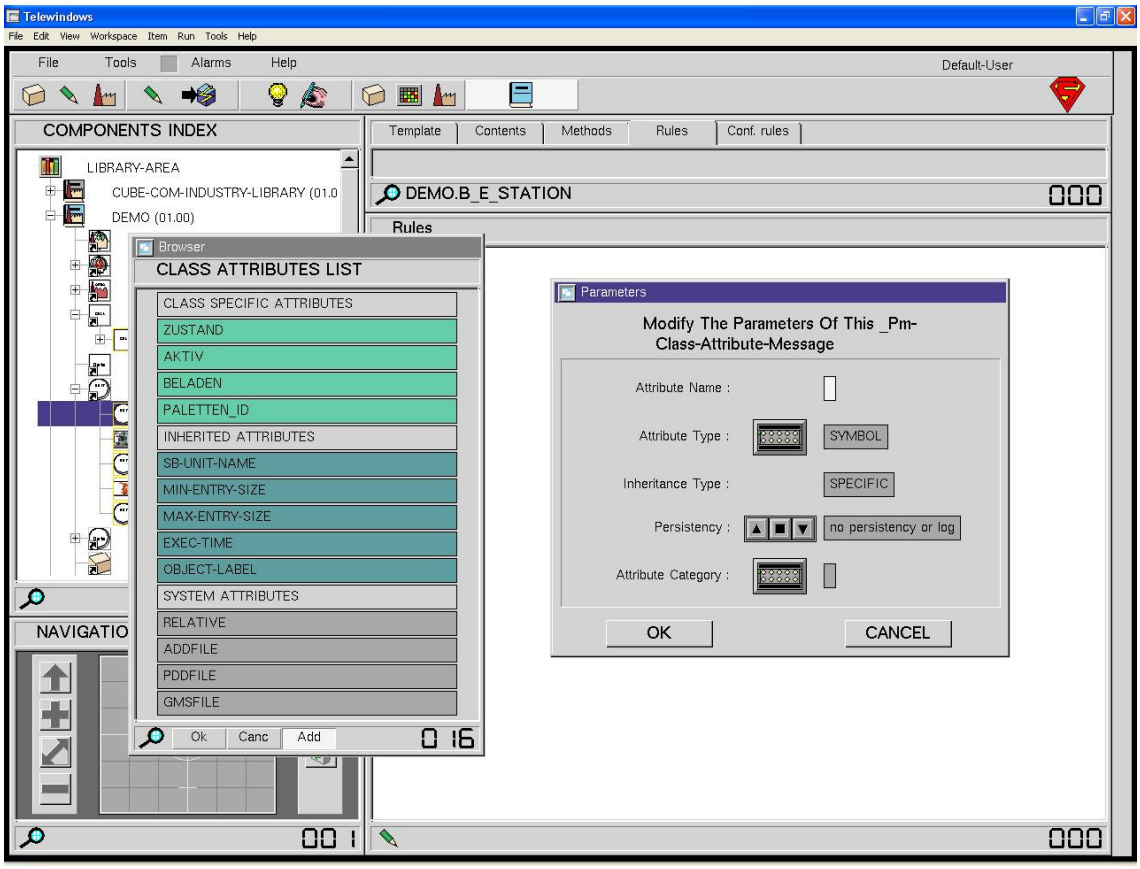

Abbildung 7.4 Hinzufügen eines Attributes

Eingaben sind nur bei „Attribute Name“ und bei „Attribute Type“ zu tätigen.

- „Attribute Name“ gibt die Bezeichnung des Attributs an.
- Unter „Attribut Type“ versteht man, welchen Datentyp das Attribut besitzen soll. In diesem Projekt wird immer eine Cube-Variable erstellt. Eine solche Variable ermöglicht die Verknüpfung mit einem Wert aus dem Anlagenbetrieb über den Real Time Data Server. Eine Cube-Variable kann jeden Datentyp annehmen.

Somit ist das Modellieren der Unit abgeschlossen.

- Unit-2 (COM)

  Die Unit „COM“ ist keine normale Unit sondern eine Logical-Unit. Über diese Unit läuft die gesamte Kommunikation mit der Anlage. Sie stellt durch ihre Methoden und Events die Verbindung über einen COM-Wrapper mit der Anlage her. Dieser COM-Wrapper ist eine im System registrierte DLL-Datei. Diese stellt Methoden zur Verfügung, welche im Production Modeler durch einen Method-Caller aufgerufen werden können. Näheres zum COM-Wrapper ist in Kapitel 10 zu finden.

  Um eine „COM“-Unit erstellen zu können, muss zuvor eine neue Bibliothek geladen werden. Diese Bibliothek muss danach noch als Subbibliothek der „DEMO“-Library definiert werden. Diese COM-Bibliothek beinhaltet ein so genanntes „COM-

LOGICAL-UNIT“, das eine Verbindung mit dem COM-Wrapper bzw. der Anlage ermöglicht. Zum Laden der neuen Bibliothek klickt man in der Library-Baumstruktur mit der rechten Maustaste auf den Stamm und wählt „Load“ aus (Abbildung 7.3). Es öffnet sich nun das Fenster „Library List“. In diesem Fenster wählt man die „CUBE-COM-INDUSTRY-LIBRARY“ aus und klickt auf „OK“.

Als nächsten Schritt muss man die eben geladene Bibliothek als Subbibliothek der „DEMO“-Bibliothek definieren. Dies geschieht, indem man nach einem Rechtsklick auf die „DEMO“-Bibliothek wieder die „CUBE-COM-INDUSTRY-LIBRARY“ auswählt und mit „OK“ bestätigt.

Sind die zuvor beschriebenen Schritte erledigt, erscheint in der Baumstruktur unter DEMO – LOGICAL-UNIT die COM-LOGICAL-UNIT. Dies ist in Abbildung 7.5 zu sehen. Um nun eine neue Unit zu erzeugen klickt man in der Baumstruktur das COM-LOGICAL-UNIT mit der rechen Maustaste und wählt „Create-Subclass“ aus. Es öffnet sich nun ein Fenster, in dem man im Feld „Name“ die Bezeichnung „COM“ eingibt. Die „COM“-Unit ist somit erstellt.

Die eben erstellte neue Klasse weiß aber noch nicht mit welchem COM-Wrapper sie arbeiten soll. Um dies festzulegen, klickt man in der Registerkarte „Template“ auf das Objekt und wählt „Parameters“ aus (Abbildung 7.5). Es erscheint dann das Fenster in Abbildung 7.6.

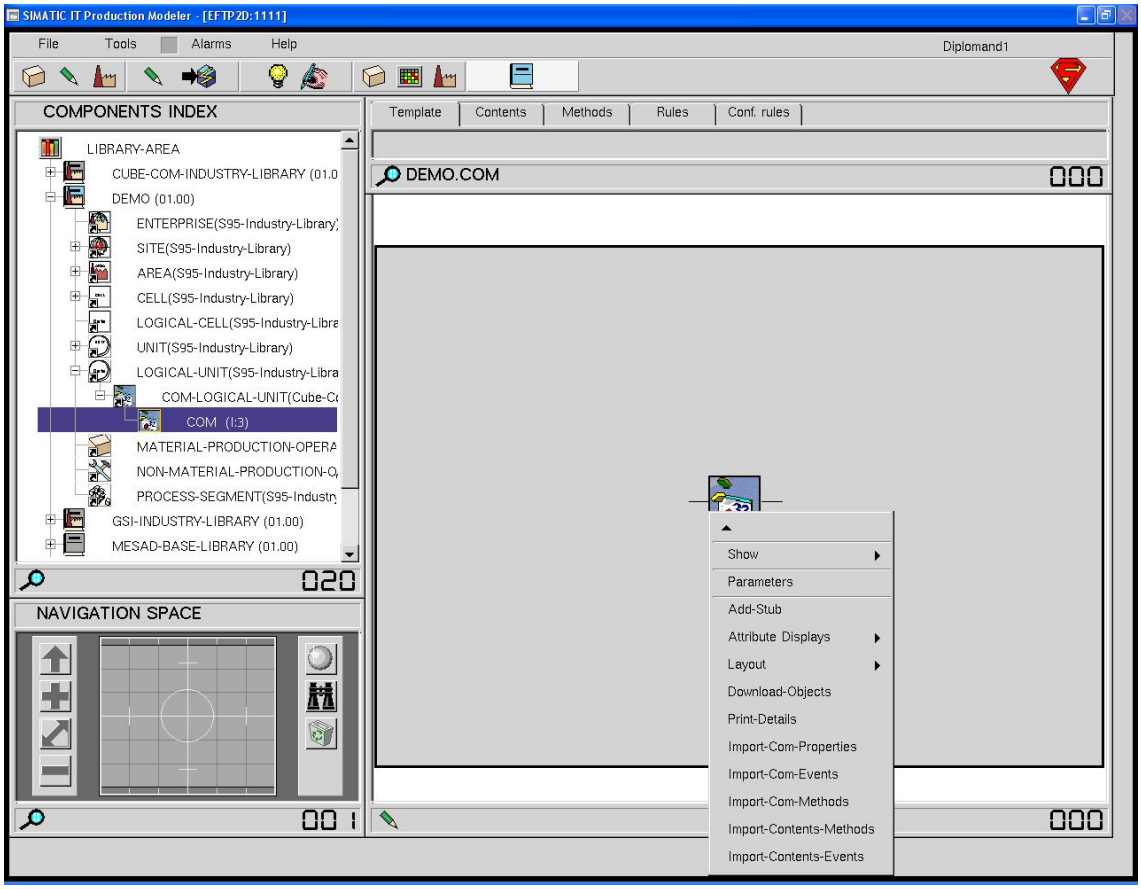

Abbildung 7.5: COM-Unit

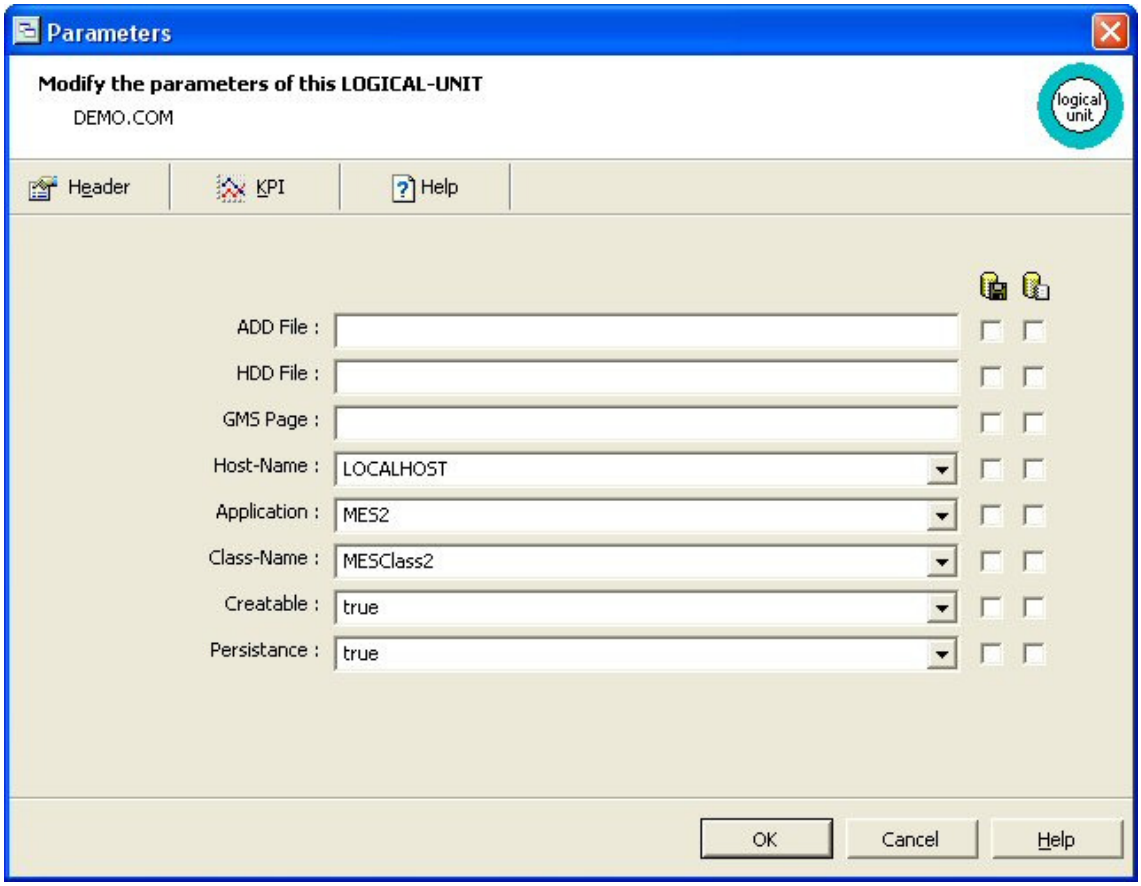

Abbildung 7.6: Parameterfenster der „COM“-Unit

Es sind nun Eingaben über die Drop-Down-Listenfelder zu tätigen:

- Host-Name: LOCALHOST (Adresse des Rechners, auf dem die DLL-Datei registriert ist)
- Application: MES2 (Bezeichnung der DLL-Datei)
- Class-Name: MESClass2

Abschließend müssen noch Methoden und Events für die Unit importiert werden. Dies geschieht wieder mit einem Klick auf das „COM“-Objekt in der Registerkarte „Template“. Dort wählt man, wie in Abbildung 7.5 zu sehen, „Import-Com-Events“ aus. Es öffnet sich nun ein Fenster, in welchem man auf das Feld „Select“ (um alle Felder auszuwählen) und anschließend auf „OK“ klickt. Diese Vorgehensweise wird mit „Import-Com-Methods“ wiederholt.

Die COM-Unit ist somit erstellt und bereit, in der Zelle genutzt zu werden.

- Unit-3 (FERTIGTEILPUFFER und ROHTEILPUFFER)

  Die Unit „PUFFER“ wird in der Anlage für den Rohteilpuffer und den Fertigteilpuffer verwendet. Die Attribute sind die gleichen, wie die der Be-/Entladestation und der Bearbeitungsstation.

- Unit-4 (MASCHINE)

  Die Unit „MASCHINE“ besitzt die beiden Attribute „BELADEN“ und „AKTIV“.

- Unit-5 (ROBOTER)

  Der Roboter hat ebenfalls „AKTIV“ und „BELADEN“.

- Unit-6 (UMSETZER_1 und UMSETZER_2)

  Der Umsetzer hat nur das Attribut „AKTIV“.

7.3.3 Erstellen der Cell

Nachdem alle Units modelliert worden sind, kann man damit beginnen die Zelle zu erstellen. Man klickt dazu in der Baumstruktur unter „DEMO“-Bibliothek mit der rechten Maustaste auf „CELL“ und wählt „Create Subclass“ aus. Beim Namen gibt man „DEMOANLAGE“ ein und klickt „OK“.

7.3.3.1 Units in die Zelle laden

In der Registerkarte „Contents“ der Zelle werden nun die zuvor erstellten Units hereingeladen. Dies geschieht, indem man auf die Schaltfläche „Hinzufügen von Elementen“ (runde Schaltfläche im Navigationsbereich des PM) klickt. Die weitere Vorgehensweise beim Hereinladen der einzelnen Units ist in der Hilfe des Production Modelers gut beschrieben und deshalb hier auch nicht näher erklärt.

7.3.3.2 Methoden und deren Erstellung

Die Zelle „DEMOANLANGE“ kann verschiedene Aktionen ausführen. Ein Beispiel für eine solche Aktion wäre der Transport einer Palette von der Be-/Entladestation zum Rohteilpuffer. Jede dieser Aktionen ist eine so genannte Methode der Zelle. Die in diesem Projekt eingesetzten Methoden dienen dazu, Regeln aufzurufen. Es muss aber nicht für jede Regel eine Methode erstellt werden, da man manche Regeln auch direkt aus der CAB-Benutzeroberfläche aktivieren kann.

Zum Hinzufügen einer Methode geht man in der Zelle „DEMOANLAGE“ auf die Registerkarte „Methods“ und klickt im unteren Teil auf „Add“. Es wird dann, statt der Baumstruktur, die „DATA-SYSTEM PALETTE“ angezeigt (Abbildung 7.7). Um eine neue Methode zu erstellen, zieht man eine „Action-Method-Definition“ in das „Methods“-Fenster des Arbeitsbereiches. Es öffnet sich dann ein Fenster, in dem man folgende Eingaben tätigt:

- Name: Hier gibt man eine Bezeichnung ein, wie zum Beispiel „ALARM“.
- Description: Optional kann man hier eine Erklärung eingeben.
- Type: Der Typ wird immer auf „CLASS-BASED“ gesetzt.
- Task-Type: Da mit einer Methode Regeln aufgerufen werden, wird dieser Parameter immer auf „RULE-TASK“ gesetzt

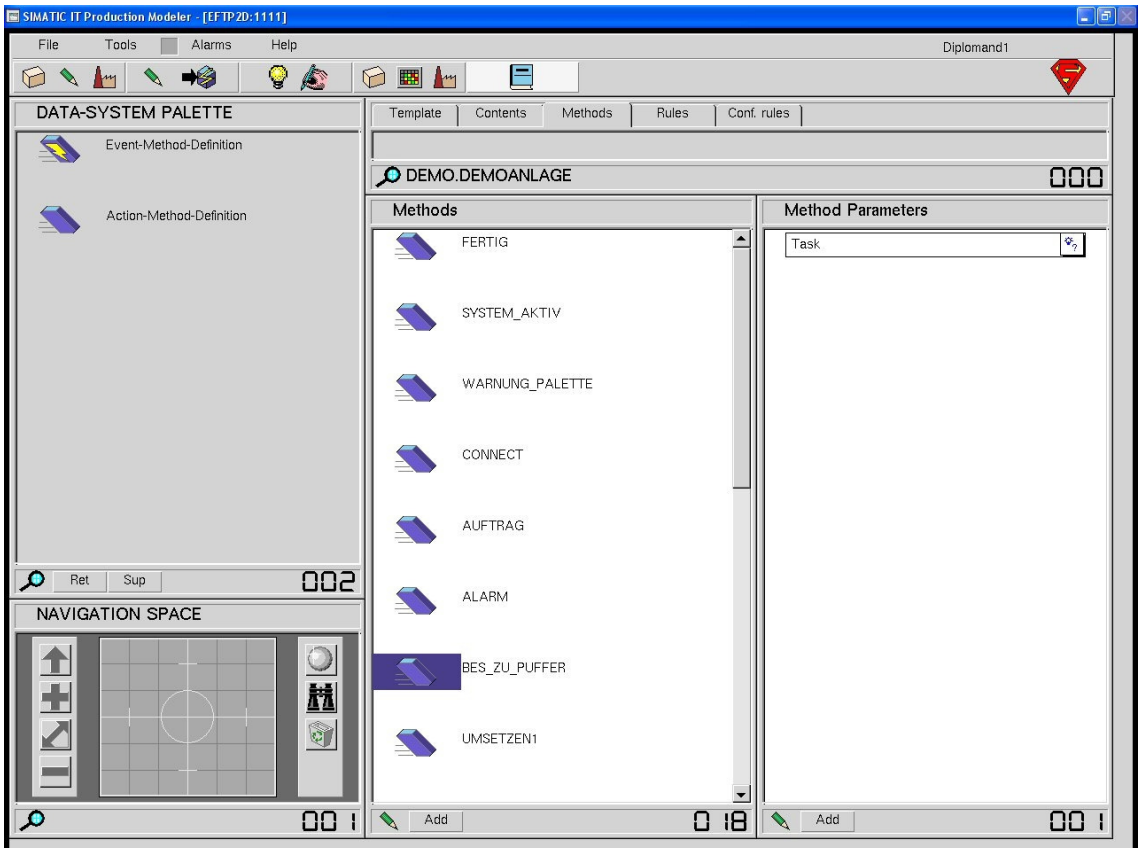

Abbildung 7.7: Erstellen einer Methode

Die Methoden müssen nach deren Erstellung noch mit den dazugehörigen Regeln verknüpft werden, die sie aufrufen sollen. Dies geschieht, indem man im Fenster „Method Parameters“ der jeweiligen Methode auf „Task“ klickt und im sich öffnenden Fenster die entsprechende Regel auswählt.

7.3.3.3 Events und deren Erstellung

Events werden benötigt, um zu signalisieren, dass ein bestimmtes Ereignis eingetreten ist. Ein Beispiel dafür ist das Fertigmelden eines abgeschlossenen Palettentransports. In diesem Projekt werden sie auch für die Fertigmeldungen bzw. für eine Alarmmeldung der Anlage verwendet. Beim Erstellen geht man ähnlich vor wie bei den Methoden. In der Registerkarte „Methods“ klickt man wieder auf den „Add“-Button. Man zieht diesmal aber ein „Event-Method-Definition“-Element in den Arbeitsbereich. Es sind außer „Name“ und „Description“ auch noch folgende Eingaben zu tätigen:

Type: `INSTANCE-BASED`

Event-Type: `EVENT-BROKER`

Bei den Events sind außer den bisherigen Einstellungen keine weiteren erforderlich.

7.3.4 Erstellen einer Area

In der Area ist nur die Zelle „DEMOANLAGE“ enthalten. Die Area besitzt keine speziellen Attribute, Methoden oder Events. Die Erstellung und das Hereinladen der Zelle funktioniert wie bei den Units und der Zelle zuvor. Aus diesem Grund ist das hier auch nicht mehr näher beschrieben.

7.4 Erstellen der Anlagenregeln im Bibliotheksbereich

Die Modellierung der Prozesse geschieht in Form von Regeln im Bibliotheksbereich des PM. Im folgenden Teil wird der Ablauf beim Erstellen einer solchen Regel von Grund auf erklärt. Unsere Demoanlage wurde, wie zuvor beschrieben, als Zelle modelliert und deshalb werden die zugehörigen Regeln auch in dieser Zelle hinterlegt.

7.4.1 Vorbereitungen zur Regelerstellung

Bevor eine eigentliche Regel erstellt werden kann, muss ein „Repac-Leaf“ konfiguriert werden. Ein solches „Repac-Leaf“ kann als Ordner angesehen werden, in dem Regeln abgelegt werden. Ab einer gewissen Anzahl von Regeln ist es von Vorteil, mehrere „Repac-Leafs“ zu erstellen, in dem dann zusammengehörige Regeln abgelegt werden können. Beispiele sind die „Repac Leafs“ Auftragsmanagement oder Automatikbetrieb in der Demoanlage.

Um ein neues „Repac-Leaf“ zu erstellen, navigiert man in der Baumstruktur des Bibliotheksbereichs zur gewünschten Klasse. In unserem Fall ist das die Zelle „Demoanlage“. Dort wählt man dann die Registerkarte „Rules“ aus.

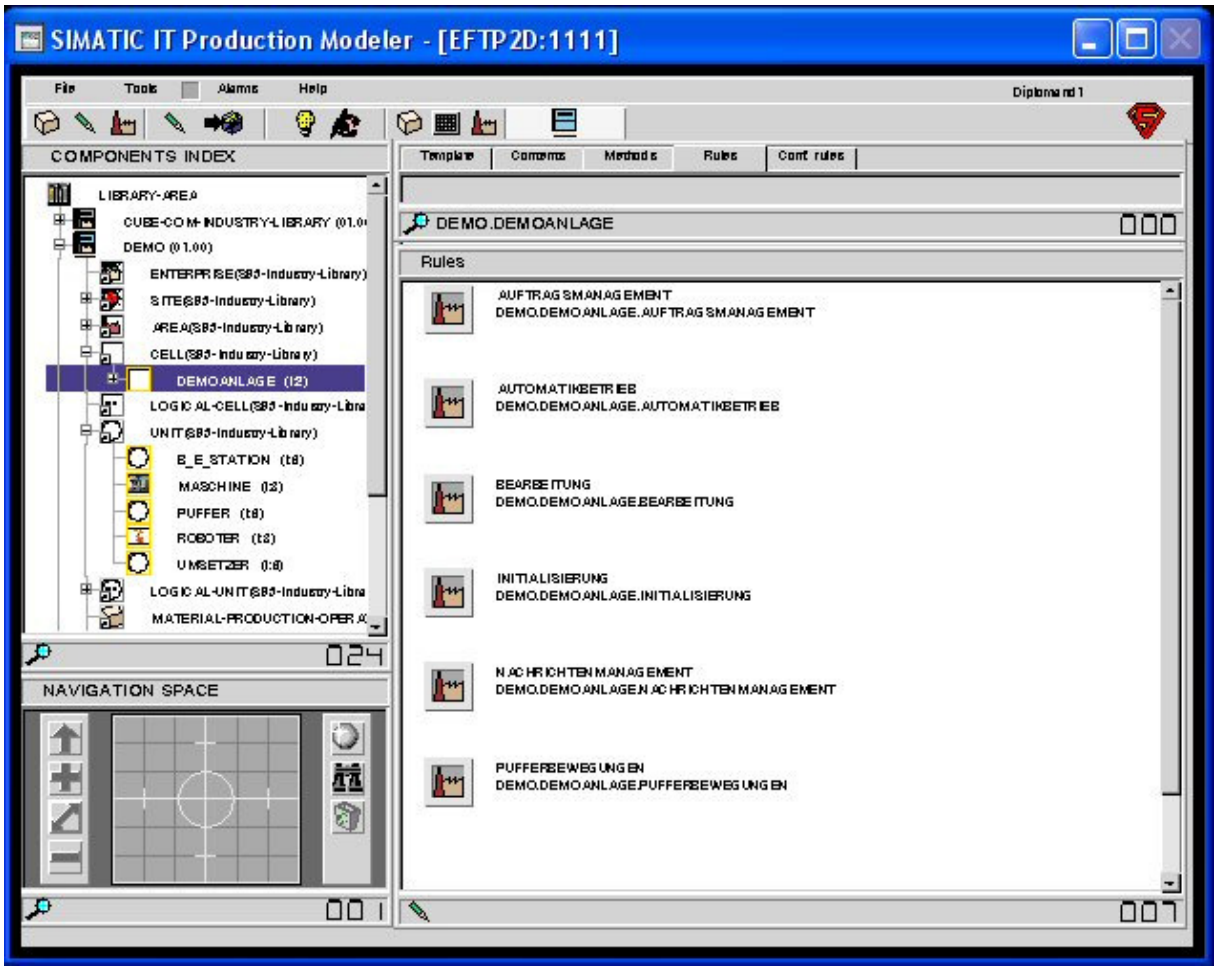

Abbildung 7.8: Registerkarte "Rules" der Demoanlage

Jetzt klickt man im Navigationsbereich auf die Schaltfläche „Hinzufügen von Objekten“. Es öffnet sich nun die „REPAC BRANCH PALETTE“. In dieser Palette klickt man auf das Element „Repac-Leaf“ und zieht es irgendwo in den Arbeitsbereich. Dort klickt man erneut. Vorsicht, es gibt kein „Drag and Drop“ wie in Windows. Es öffnet sich dann das Fenster “Parameters“ (Abbildung 7.10).

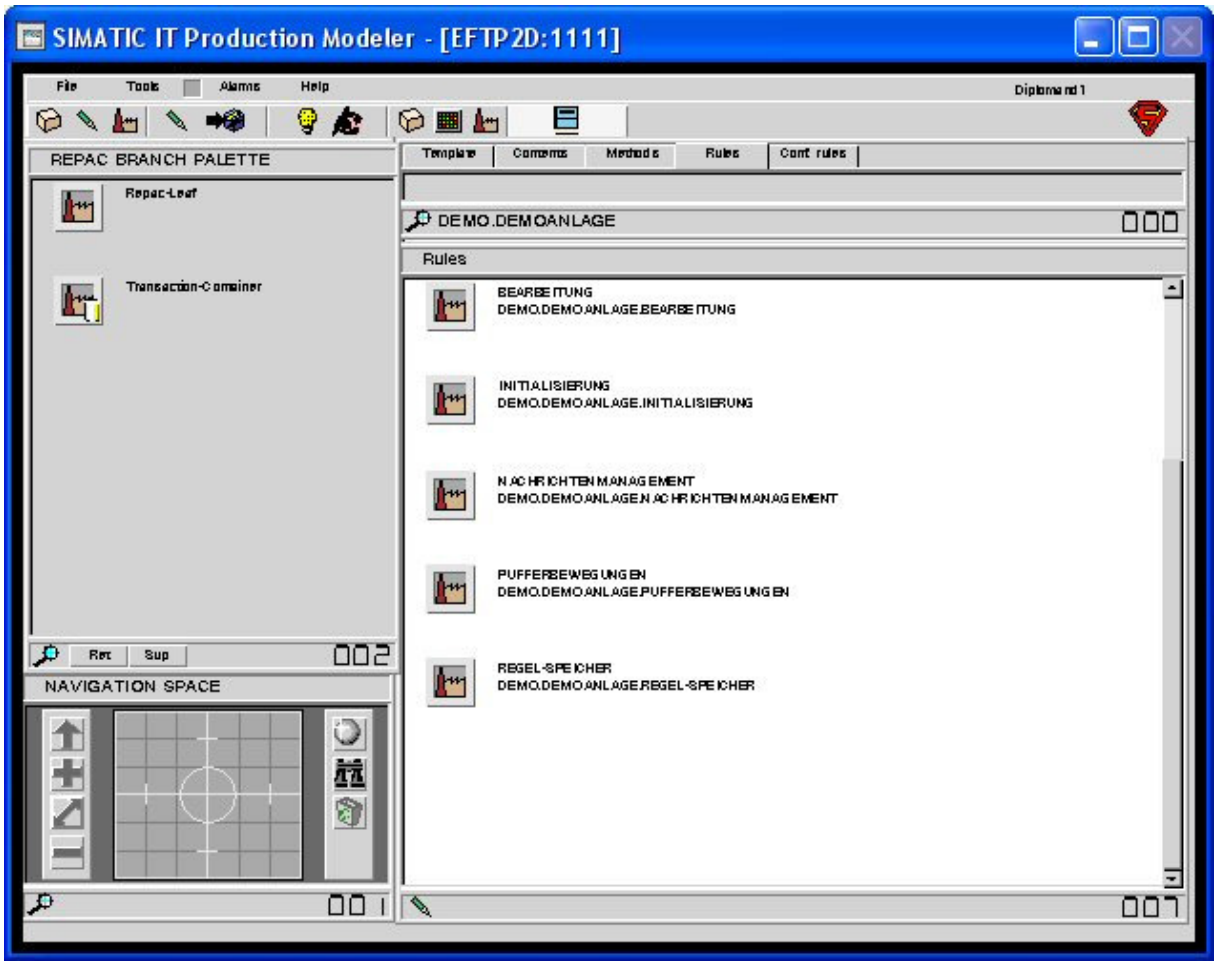

Abbildung 7.9: „REPAC BRANCH PALETTE“

Im Fenster „Parameter“ können Name und Regeltyp festgelegt werden. In diesem Fall heißt das „Repac-Leaf“ „AUFTRAGSMANAGEMENT“ und der Regeltyp ist „INSTANCE-BASED“. Näheres zum Unterschied zwischen klassen- und instanzbasiert findet man in der SIMATIC IT Production Suite Literatur. Die „Repac-Category“ kann hier nicht beeinflusst werden. Nach Eingabe der Daten wird mit der „OK“-Taste bestätigt.

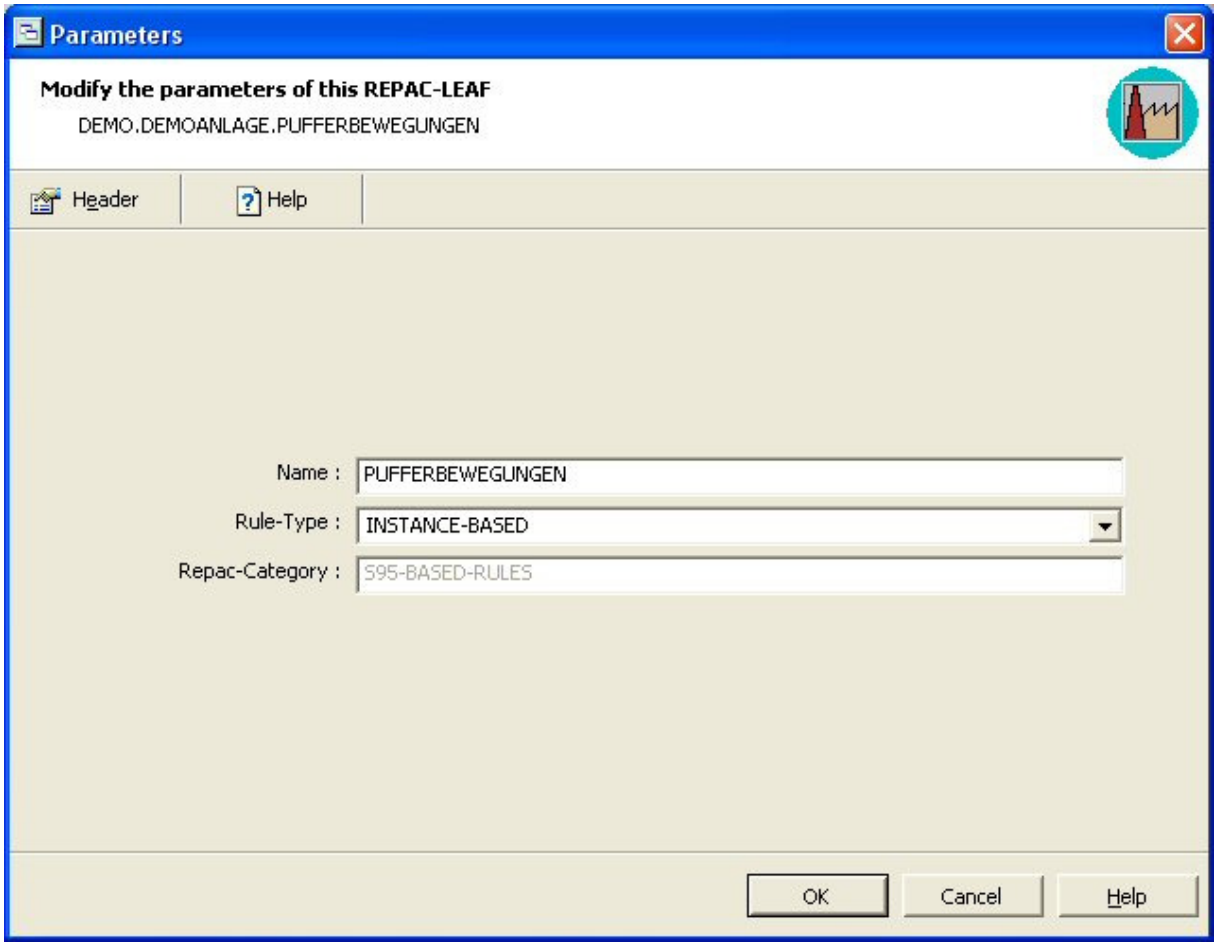

Abbildung 7.10: „REPAC-LEAF“ Parameterfenster

Zum Öffnen des neuen „Repac-Leafs“ gibt es zwei Möglichkeiten. Man kann entweder in der Baumstruktur die Klasse „Demoanlage“ expandieren und dort das gewünschte „Repac-Leaf“ anwählen, oder man klickt im Arbeitsbereich mit der rechten Maustaste auf das gewünschte „Repac-Leaf“ und wählt dort den Befehl „Open“ aus. Beides funktioniert nur, wenn die Registerkarte „Rules“ der Zelle ausgewählt ist.

7.4.2 Erklärung der Elemente in der Regelpalette

Eine Regel wird aus einzelnen Komponenten erstellt. Im folgenden Teil werden die wichtigsten Komponenten genauer beschrieben. Zuerst wird auf die Handhabung solcher Objekte eingegangen und Eigenschaften erklärt, die jedes dieser Objekte besitzt. Danach wird genauer auf die einzelnen Komponenten eingegangen.

7.4.2.1 Allgemeines zu den Elementen

Die Regelelemente befinden sich in einer Palette, die mit der Schaltfläche „Hinzufügen von Elementen“ geöffnet werden kann. Es ist zu beachten, dass man sich beim Drücken der Schaltfläche gerade in einem Regelordner („Repac-Leaf“) befindet.

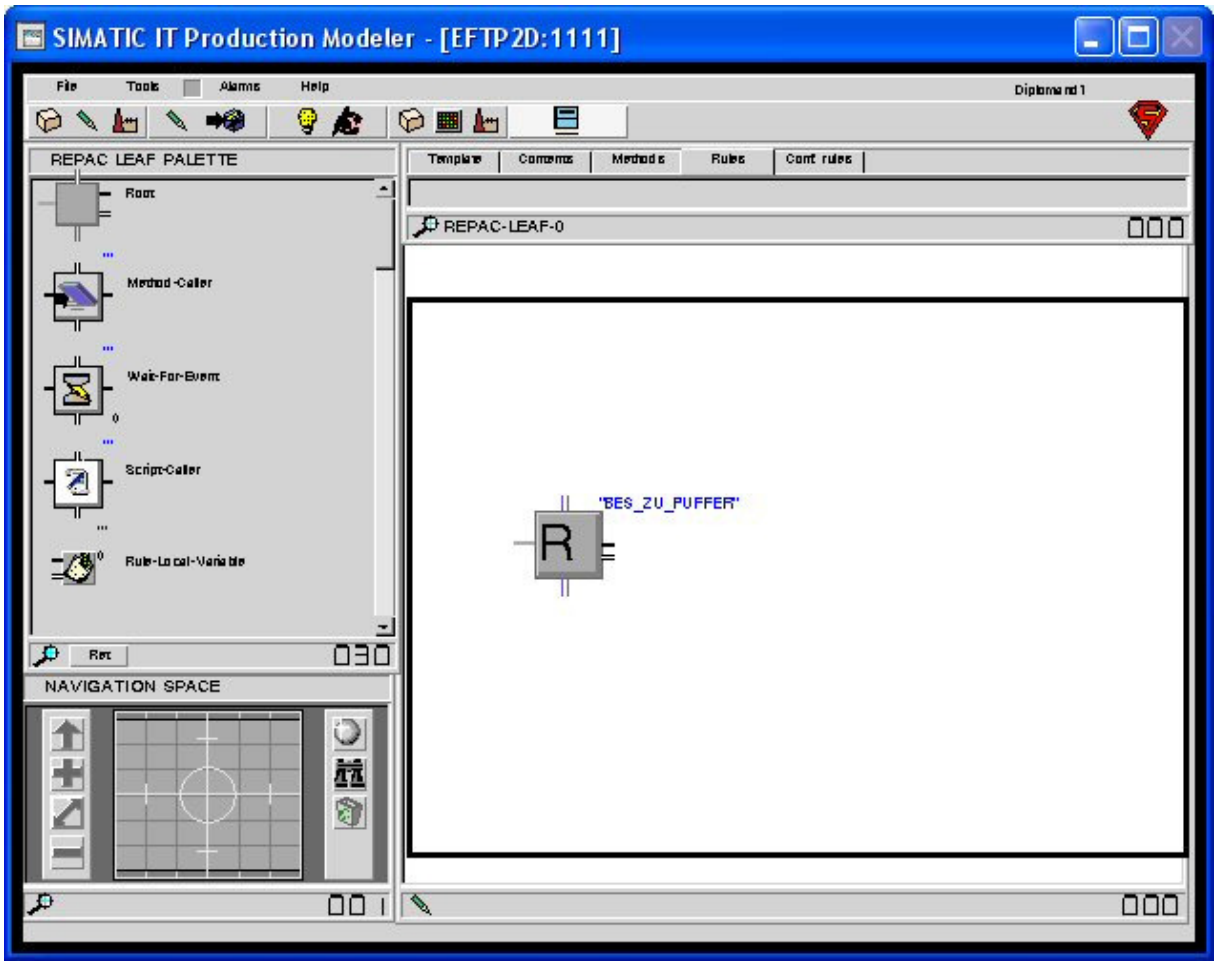

Abbildung 7.11: „REPAC LEAF PALETTE“

- Stränge

  Jedes Element hat Verbinder, mit denen die Verbindung zu einem anderen Element hergestellt wird. Verbinder, die von links oder oben kommen sind immer Eingänge. Gehen sie rechts oder unten weg, sind es Ausgänge. Ein schwarzer Verbinder ist ein positiver Aus- oder Eingang. Ein grauer Verbinder mit einem Doppelstrich ist ein negativer Aus- oder Eingang.

- Parameters

  Jedes Element in dieser Palette besitzt ein Fenster, das „Parameters“ heißt. Dieses Fenster öffnet sich beim ersten Platzieren eines Elements im Arbeitsbereich von selbst oder indem man das Symbol im Arbeitsbereich anklickt und anschließend den Punkt „Parameters“ auswählt. In diesem Fenster werden die Haupteinstellungen eines jeden Elementes eingegeben. Als Beispiel dazu ist das Parameterfenster eines „Method-Callers“ in Abbildung 7.12 zu sehen. Bei allen Elementen ist zu beachten, dass die Bezeichnung des Feldes „Name“ (manchmal auch „ID“) in einem Regelordner nur einmal vorkommen darf. Manchmal gibt es auch ein Feld „History Keeping“. Dieses Feld kann ignoriert werden, da es für eine zukünftige Version von

SIMATIC IT vorgesehen ist. Die weiteren Felder werden dann bei der Beschreibung der einzelnen Symbole genauer erklärt.

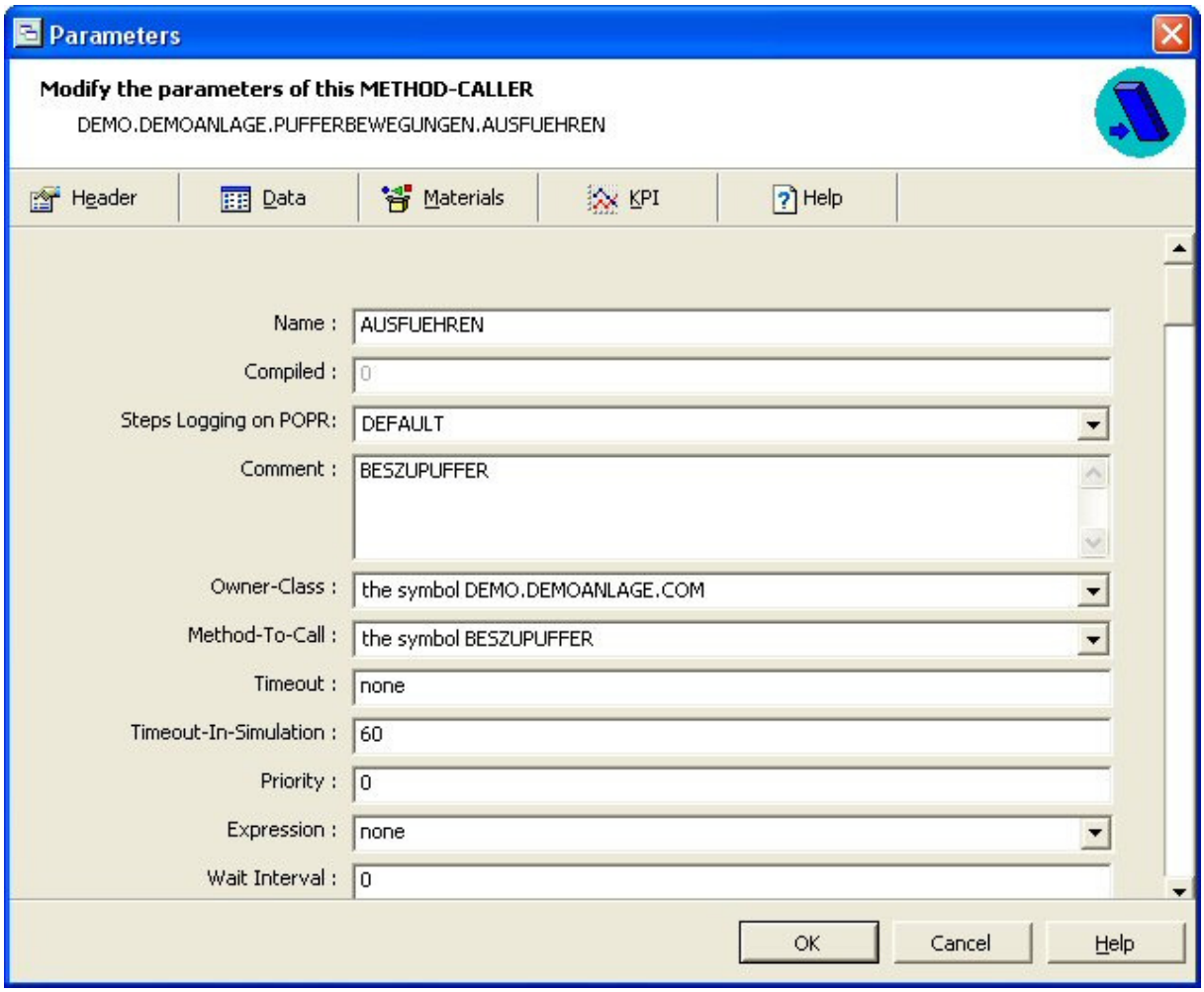

Abbildung 7.12: Parameterfenster eines „Method-Callers“

- Arguments:

  Viele Elemente in der Regelordner-Palette haben auch eine Eigenschaft „Arguments“. In diesem Fenster kann man Ein- und Ausgabeargumente für das ausgewählte Element festlegen. Eingabeargumente sind Informationen, die das Element vor der Ausführung erhält. Ausgabeargumente werden nach der Ausführung ausgegeben.

  Zu diesen „Arguments“ gelangt man, wenn man im Fenster „Parameters“ die Registerkarte „Data“ auswählt (Abbildung 7.13). Es gibt noch die Möglichkeit, das Parameterfenster mit einem Klick auf das jeweilige Symbol im Arbeitsbereich zu öffnen. Diese Art des Öffnens sei aber nur am Rande angemerkt, da das Fenster ein anderes Layout hat und nicht so bedienerfreundlich ist. Das weitere Vorgehen mit dieser Methode ist auch nicht näher beschrieben.

Will man ein bestehendes Argument ändern, so klickt man es mit der rechten Maustaste an und wählt „Edit“ aus. Um ein neues Argument zu erstellen, klickt man mit der rechten Maustaste ins Input- oder Outputfeld und wählt dann den Befehl „New“ aus.

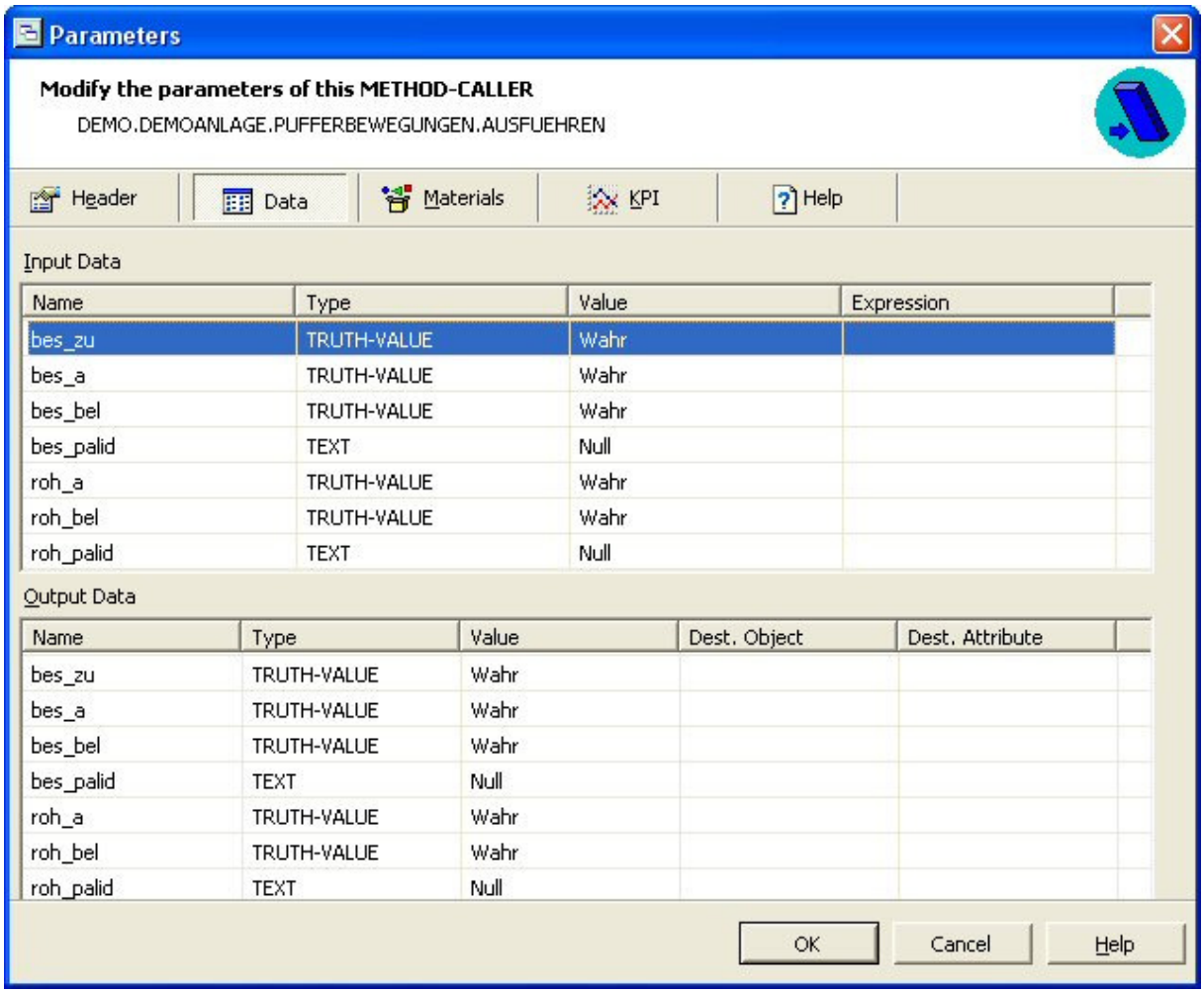

Abbildung 7.13: Attribute eines „Method-Callers“

Hat man den Befehl „New“ ausgewählt, so ändert sich die Ansicht. Im Abbildung 7.14 ist die Eingabemaske für ein Inputargument dargestellt.

- Inputargument

  Name: In dieses Feld muss eine individuelle Bezeichnung des Arguments eingegeben werden.

  Description: Nach Wunsch kann man hier eine Erklärung zum jeweiligen Element hinterlegen.

  Type: Hier wird ausgewählt, welchen Datentyp das Argument haben soll.

  Default-Value: Wenn das jeweilige Argument einen fixen Wert haben soll, wird dieser hier eingegeben.

Expression: Will man dem Argument eine Variable oder ein Output-Argument eines anderen Elements zuweisen, geschieht das in diesem Eingabefeld. Berechnungen und andere Funktionen, wie zum Beispiel das Beschneiden von Text oder das Vergleichen von zwei Größen, finden auch in diesem Feld statt.

- Outputargument:

  Die Felder „Name“, „Description“, „Type“ und „Default-Value“ sind gleich wie bei einem Inputargument.

  Destination Object: Hier kann man das Output-Argument direkt einer Variable zuweisen oder man kann ein anderes Objekt (z.B. einen Method-Caller) auswählen.

  Destination Attribute: Hat man beim „Destination Object“ ein anderes Objekt und keine Variable ausgewählt, legt man hier das Attribut fest, auf das man die Information schreiben will.

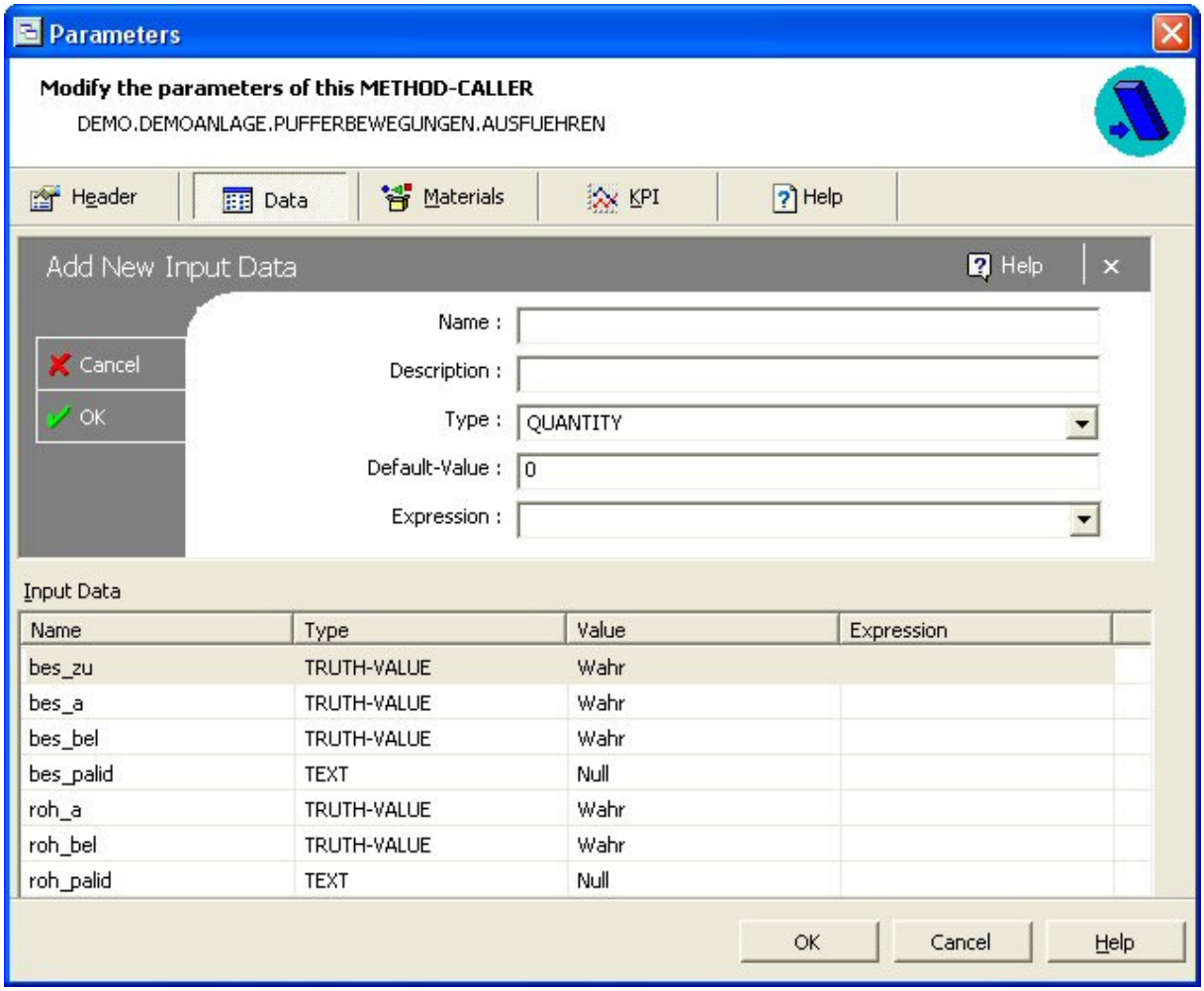

Abbildung 7.14: Erstellen eines neuen Attributes

7.4.2.2 Erklärung der einzelnen Objekte

Hier werden die Eigenschaften der einzelnen Regelbestandteile erklärt.

- ROOT

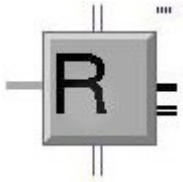

Abbildung 7.15: ROOT

Das ROOT-Symbol ist das Startelement jeder Regel. In einem leeren Regelordner befindet sich immer ein ROOT-Symbol, mit dem man gleich eine Regel beginnen kann. Will man weitere Regeln im „Repac-Leaf“ erstellen, so zieht man ein ROOT-Symbol aus der „Repac-Leaf“-Palette. Im Parameterfenster werden normalerweise nur „Name“ bzw. „Description“ eingegeben. Bei den Argumenten kann man Ein- oder Ausgabeargumente definieren.

- Method-Caller

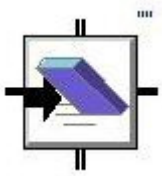

Abbildung 7.16 Method-Caller

Ein Method-Caller ist ein sehr wichtiges Element beim Modellieren von Regeln. Der Method-Caller ruft Methoden auf, die zuvor in der Registerkarte „Methods“ oder in einem „COM“-Objekt hinterlegt wurden. Diese Methoden rufen dann neue Regeln auf oder kommunizieren über „COM“ mit der Anlage. Das Objekt bleibt solange aktiv, bis die aufgerufene Regel oder der Vorgang an der Anlage beendet ist. Neben einer Beschreibung und Benennung sind in der Registerkarte „Header“ des Parameterfensters folgende Eingaben wichtig:

- Owner-Class: Dies gibt an, wo die aufzurufende Methode zu finden ist. Hier ist das immer die Zelle „DEMOANLAGE“, welche man in der Drop-Down-Liste auswählen kann.
- Method-To-Call: Hat man eine „Owner-Class“ festgelegt, kann man in der Drop-Down-Liste dieses Eingabefeldes die gewünschte Methode auswählen.

Eine nähere Beschreibung zu den Methoden und deren Erstellung findet man in Punkt 7.3.3.2.

In der Registerkarte „Data“ können Input- und Outputargumente definiert werden. Diese Argumente werden dann an die aufgerufene Methode übergeben bzw. von der Methode übernommen.

- GSI-Method-Caller

Abbildung 7.17: GSI-Method-Caller

Die GSI-Method-Caller sind ebenfalls sehr wertvolle Elemente bei der Regelerstellung. Es können damit Methoden aufgerufen werden, die von den SIMATIC IT Komponenten zur Verfügung gestellt werden. Ein Beispiel dafür ist die Methode „AddHut“, die im Material Manager eine neue Handling Unit erstellt. Die wichtigsten Komponenten, mit denen kommuniziert werden kann, sind der Production Order Manager und der Material Manager. In Abbildung 7.17 ist ein GSI-Method-Caller für den Material Manager abgebildet.

Im Parameterfenster ist neben Bezeichnung und Beschreibung die „Method-to-Call“ auszuwählen. Diese Methode beschreibt die Anwendung, die in der gewünschten Komponente ausgeführt werden soll, und ist in der Drop-Down-Liste auszuwählen.

In der Registerkarte „Data“ werden nach dem Auswählen der Methode anwendungsspezifische Argumente angezeigt. Welche ausgefüllt werden müssen und welche nicht, wird in der SIMATIC IT Hilfe beschrieben.

Die GSI-Method-Caller sind den System-Method-Callern vorzuziehen, da die GSI-Method-Caller schneller arbeiten und auch mehr Anwendungen der SIMATIC IT Komponenten zu Verfügung stellen.

- Event-Sender

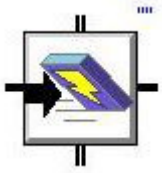

Abbildung 7.18: Event-Sender

Ein Event-Sender ist dafür zuständig, ein Signal auszulösen. Neben Name und Beschreibung sind eine „Owner-Class“ und eine „Method-to-Call“ einzugeben. Diese beschreiben, wo der Event zu finden ist und welcher Event gesendet werden soll.

- Wait-for-Event

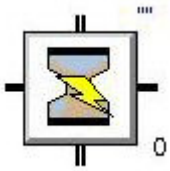

Abbildung 7.19 Wait-for-Event

Ein „Wait-for-Event“ ist ein Objekt, das nach der Aktivierung so lange wartet, bis ein bestimmtes Signal (Event) gesendet wird. Erst nach Einlangen des Signals wird die Regel weiter ausgeführt. Im Parameterfenster wird wieder eine Bezeichnung und Beschreibung eingegeben. Außerdem sind eine „Owner-Class“ und ein „Event-to-Wait-for“ festgelegt werden.

- Owner-Class: Hier wird festgelegt, wo der Event zu finden ist. In diesem Projekt gibt es Events nur in der Zelle, die Eingabe lautet immer „the symbol DEMO.DEMOANLAGE“. Dies kann aus einer Drop-Down-Liste ausgewählt werden.
- Event-to-Wait-for: Dieses Eingabefeld legt fest, auf welchen Event gewartet werden soll. Alle möglichen Events werden in der Drop-Down-Liste angezeigt.

Näheres zu Events und zum Erstellen dieser ist in Punkt 7.3.3.2 zu finden.

- Send-Message

Abbildung 7.20: Send-Message

Ein Send-Message-Objekt ruft eine im Template Editor des Messaging Managers vorbereitetes Bildschirmfenster auf. Diese Meldung kann auch eine Eingabemaske für bestimmte Werte besitzen.

Neben Bezeichnung und Beschreibung sind im Parameterfenster folgende Eingaben wichtig:

- Destination: Dies legt fest, wo das Template liegt

- Call-type: Dieser soll immer auf „CALL“ gesetzt sein.
- Message-Type: Hier wird „UNSOLICITED“ ausgewählt. Damit taucht die Meldung als eigenständiges Fenster auf und muss nicht über das Display des Messaging Managers laufen.
- Message-Template: Mit einem Klick auf die Schaltfläche des Eingabefeldes muss man eine vorbereitete Nachrichtenvorlage definieren.

In der Registerkarte „Data“ des Parameterfensters sind die Input- und Outputargumente festzulegen. Will man etwas Bestimmtes im Fenster anzeigen lassen, definiert man ein Inputargument. Um Werte, die in dem Nachrichtenfenster eingegeben wurden, zu erhalten, definiert man Outputargumente. Bei beiden Argumenttypen ist wichtig, dass die Namen die gleichen wie in der Nachrichtenvorlage sind. Auf Datentyp und Reihenfolge ist ebenfalls zu achten.

- Set-Variable

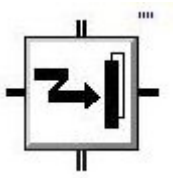

Abbildung 7.21: Set-Variable

Eine Set-Variable schreibt einen Wert, zum Beispiel in eine Rule-Local-Variable. Man kann aber auch die Argumente anderer Objekte setzen.

Neben Name und Beschreibung sind im Parameterfenster folgende Eingaben wichtig:

- Object-Name: Hier wählt man entweder gleich eine zu setzende Variable aus, oder wählt ein Objekt, dessen Attribut (Argument) gesetzt werden soll.
- Attribute-Name: Wurde bei „Object-Name“ ein Objekt und keine Variable ausgewählt, ist hier das zugehörige Argument in der Drop-Down-Liste auszuwählen.

Wurden diese Eingaben getätigt, klickt man auf „OK“ und man gelangt zur Eingabemaske „Expression“. Hier wird nun festgelegt, was geschrieben werden soll. Dabei ist darauf zu achten, dass die Datentypen übereinstimmen.

- Rule-Local-Variable

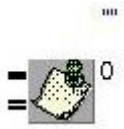

Abbildung 7.22: Rule-Local-Variable

Eine Rule-Local-Variable ermöglicht das Speichern einer Variablen, um in einer Regel als Ein- oder Ausgabeoperation genützt zu werden. Neben den Standardfeldern ist hier noch ein „Initial Value“ einzugeben. Wird hier ein Wert eingegeben, besitzt ihn die Variable solange, bis etwas anderes hineingeschrieben wird. Die Anordnung erfolgt am besten gleich hinter dem Objekt, welches die Variable setzen soll. Dies kann zum Beispiel ein Set-Variable-Objekt sein. Manchmal ist es aber auch notwendig, die Variable vor dem Objekt einzufügen, wie zum Beispiel beim Send-Message-Objekt.

- Loop und Loop-In

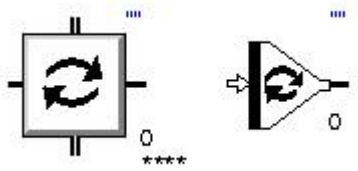

Abbildung 7.23: Schleifenelemente

In SIMATIC IT besteht eine Schleife aus einem LOOP-Objekt und einem LOOP-IN Objekt. Das LOOP-Objekt startet eine Schleife, die beliebig oft durchlaufen werden kann. Das LOOP-IN-Objekt ist das Startobjekt für die eigentliche Schleife. Beim LOOP-Objekt sind außer den Standardfeldern Name und Beschreibung noch ein „Port-Name“ und „Max-Iterations“ wichtig. Der „Port-Name“ gibt an, welches LOOP-IN Objekt zum LOOP-Objekt gehört und „Max-Iterations“ gibt die Anzahl der Durchläufe der Schleife an. Man kann hier einen Fixwert oder einen Ausdruck eingeben. Will man eine Regel endlos durchlaufen lassen, gibt man bei „Max-Iterations“ Infinite-Loop ein. Man kann auch noch eine „Parameter-List“ eingeben, um Werte in die Schleife zu übergeben. Im Loop-In-Objekt werden Name und ein Kommentar eingegeben.

- Delay

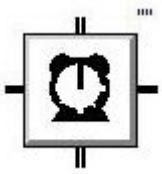

Abbildung 7.24: Delay

Benötigt man eine Zeitverzögerung, wird ein Delay-Objekt eingeführt. Man kann damit zum Beispiel eine Bearbeitungszeit simulieren. Im Parameterfenster wird im Feld „Analysis-Delay“ die Verzögerungszeit in Sekunden eingegeben. Wird ein Delay-Objekt gestartet, bleibt es so lange aktiv, bis die Verzögerungszeit abgelaufen ist. Erst dann läuft die Regel weiter.

- End-of-Rule

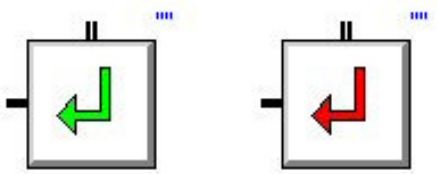

Abbildung 7.25: End-of-Rule

Das „End-of-Rule“-Symbol beendet eine Regel. Sind noch irgendwelche anderen Objekte aktiv, wie zum Beispiel ein Wait-for-Event, werden diese sofort beendet. Im Parameterfenster kann man neben Name und Beschreibung noch den „Exit-Status“ festlegen. Man kann den Status auf „OK“ oder „NOT-OK“ setzen und damit signalisieren, ob eine Regel richtig ausgeführt wurde, oder ob ein Fehler aufgetreten ist. „OK“ ist in Abbildung 7.25 links und „NOT-OK“ rechts zu sehen.

7.4.3 Erklärung der Regelerstellung anhand einer Beispielregel

In diesem Kapitel wird die Erstellung einer Regel erklärt. Als Beispielregel wird die Regel BES_ZU_PUFFER herangezogen. Die fertige Regel ist in Abbildung 7.26 dargestellt. Die Regel soll den Transport einer Palette von der Be-/Entladestation zum Rohteilpuffer durchführen.

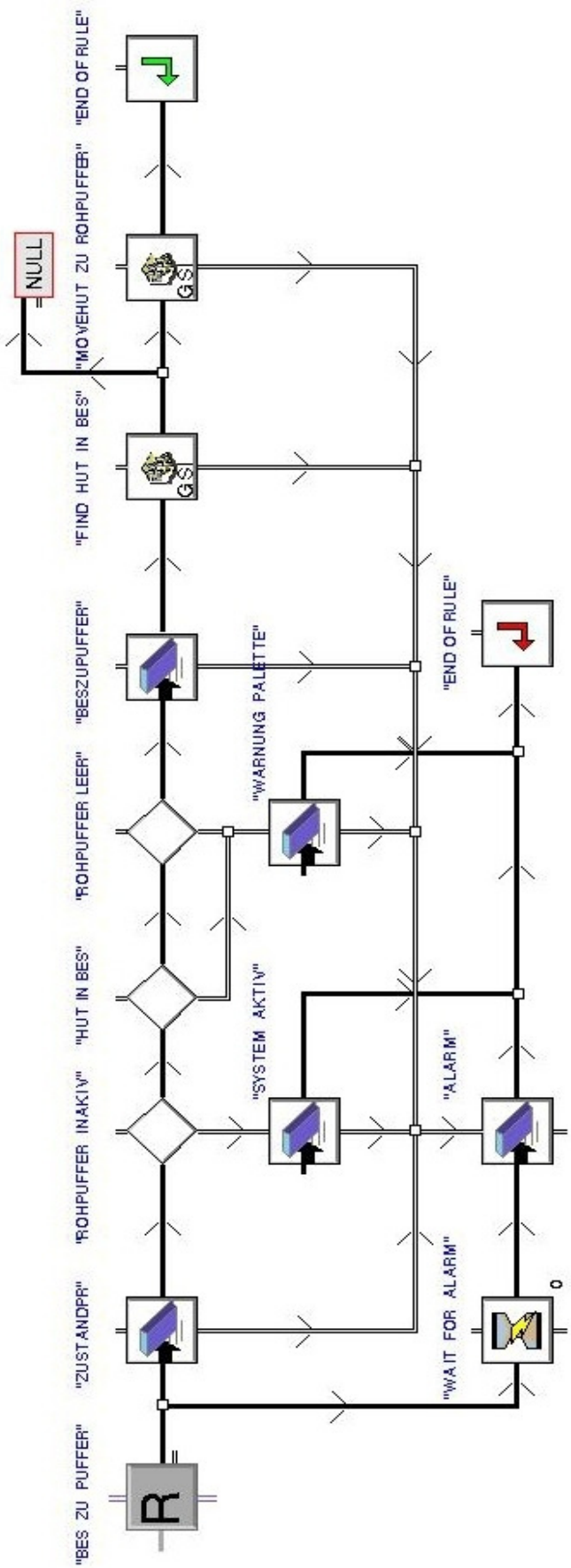

Abbildung 7.26: PM-Regel BES_ZU_PUFFER

Regelerstellung:

Als Beispiel zur Regelerklärung wird, wie bereits oben erwähnt, die Regel „BES_ZU_PUFFER“ herangezogen. Bevor man beginnen kann die Regel zu erstellen, muss man sich im Klaren sein, wie der Ablauf in Wirklichkeit aussehen soll. In diesem Fall soll eine Palette von der Be-/Entladestation zum Rohteilpuffer transportiert werden. Dies darf nur durchgeführt werden, wenn folgende Punkte erfüllt sind:

- Das Förderband des Rohteilpuffers bzw. der Be-/Entladestation darf nicht aktiv sein.
- In der Be-/Entladestation muss eine Palette liegen.
- Im Rohteilpuffer darf sich keine Palette befinden.

Diese Bedingungen werden in der realen Anlage über die SPS des Puffers abgefragt. Das MES überprüft sie in der Regel zur Sicherheit noch einmal, indem es über den Zellenrechner die aktuellen Zustände des Puffers abfragt. Sind alle Abfragen positiv, kann der „GO“-Befehl gegeben und die Bewegung durchgeführt werden. Es soll auch noch die Möglichkeit bestehen, auftretende Fehler und Störungen anzuzeigen.

Wenn die Funktionen der Regel festgelegt sind, kann man mit der Erstellung der Regel beginnen. Ist kein ROOT-Symbol vorhanden, zieht man sich eines aus der Palette in den Arbeitsbereich.

- ROOT-Symbol: Bei diesem Objekt sind nur „Name“ und „Description“ einzugeben. Argumente werden keine benötigt, da die Regel keine besonderen Eingangs- oder Ausgangsdaten besitzt.
  - Name: `BES_ZU_PUFFER`
  - Descripiton: `BES_ZU_PUFFER`
- Method-Caller „ZUSTANDPR“: Die Regel soll zur Sicherheit die Zustände der Anlage abfragen und erkennen, ob eine Ausführung der Funktion überhaupt möglich ist. Darum soll eine Methode die aktuellen Zustände der Anlage abfragen. Man muss also direkt mit der Anlage kommunizieren. Hierzu wurde in der Registerkarte „Contents“ (Inhalte) der Zelle ein eigenes „COM“-Element hinterlegt. Näheres zum Erstellen dieses Elements ist im Teil 7.3.2 zu finden. Das COM-Objekt enthält eine Anzahl von selbst programmierten Methoden. Eine dieser Methoden ist die gesuchte „ZUSTANDPR“-Methode. Man zieht nun also den Method-Caller in den Arbeitsbereich und es öffnet sich automatisch das Parameterfenster. Folgende Eingaben sind im Parameterfenster vorzunehmen:
  - Name: ZUSTANDPR
  - Comment: ZUSTANDPR
  - Owner-Class: the symbol DEMO.DEMOANLAGE.COM (Drop-Down-Liste)
  - Method-To-Call: ZUSTANDPR (Drop-Down-Liste)

  Alle anderen Felder bleiben auf der Standardeinstellung.

Durch das Auswählen der Methode wurden automatisch Argumente in der Registerkarte „Data“ hinterlegt. Jedes Argument steht für ein Attribut der realen Zelle oder Einheit. Hier sind nur die Outputargumente relevant. Die Inhalte müssen jetzt auf die softwaremäßigen Attribute der Zelle bzw. der einzelnen Einheiten geschrieben werden. Wie bei allen Argumenten vorzugehen ist, beschreibt das folgende Beispiel.

Das Argument im beschriebenen Beispiel ist „bes_zu“. Dieses Argument enthält den Zustand der Be-/Entladestation.

Um ein Output-Argument zu bearbeiten, klickt man es mit der rechten Maustaste und wählt dann „Edit“ aus. Ist das geschehen, erscheint das Fenster in Abbildung 7.27. Achtung, zum Verlassen dieses Fensters gibt es nur die zwei Schaltzeichen „Cancel“ (mit einem roten x gekennzeichnet) und „OK“ (mit einem grünen Haken gekennzeichnet).

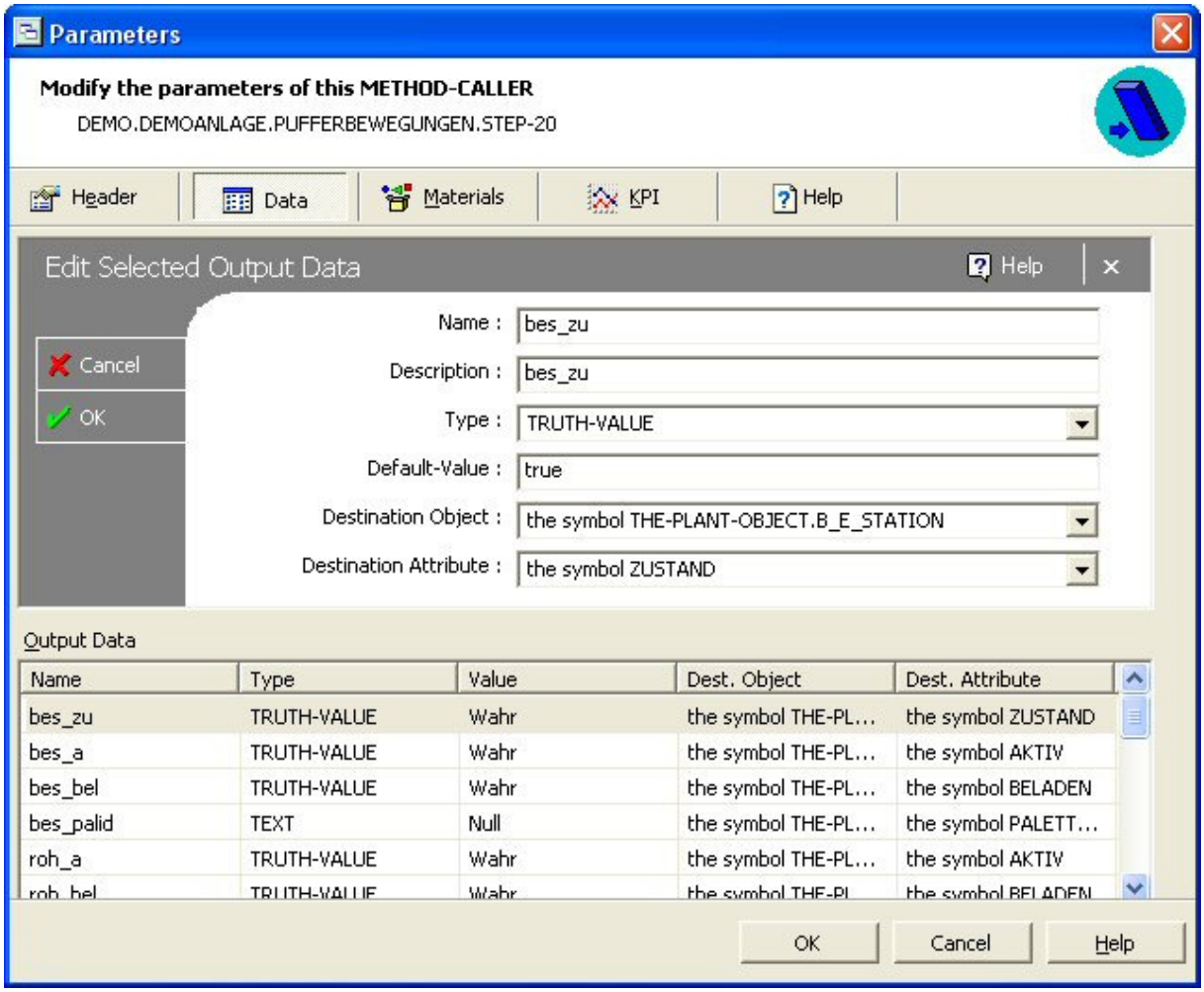

Abbildung 7.27: Fenster mit Output-Argumenten

Es sind nur die letzten beiden Felder zu bearbeiten, da die anderen bereits vorbestimmt sind. Im „Destination Object“ wird festgelegt, auf welche Zelle oder Einheit

geschrieben werden soll. In unserem Fall ist das die Be-/Entladestation. Hat man dies in der Drop-Down-Liste ausgewählt, kann man das zugehörige „Destination Attribute“ (in unserem Fall „ZUSTAND)“ ebenfalls in einer Drop-Down-Liste auswählen. Danach bestätigt man mit „OK“ und geht zum nächsten Argument.

In dieser Regel werden nicht alle Outputargumente benötigt. Man sollte aber gleich allen Argumenten das richtige „Destination Object“ und „Destination Attribute“ zuweisen, da man den Method-Caller „ZUSTANDPR“ für alle Regeln verwenden kann. Man muss also nicht für jede Regel die Argumente neu setzen, sondern kopiert einfach den Method-Caller. Dabei ist darauf zu achten, dass der Name des Method-Callers nur einmal vorkommen darf.

- Die drei Condition-Objekte: Hier werden die zur Regelausführung notwendigen Bedingungen abgefragt. Als Beispiel wird das Condition-Objekt „ROHPUFFER_INAKTIV“ durchexerziert.

  Man zieht ein Condition-Objekt aus der Palette in den Arbeitsbereich und es öffnet sich das Parameterfenster. Ein solches Fenster ist in Abbildung 7.28 zu sehen.

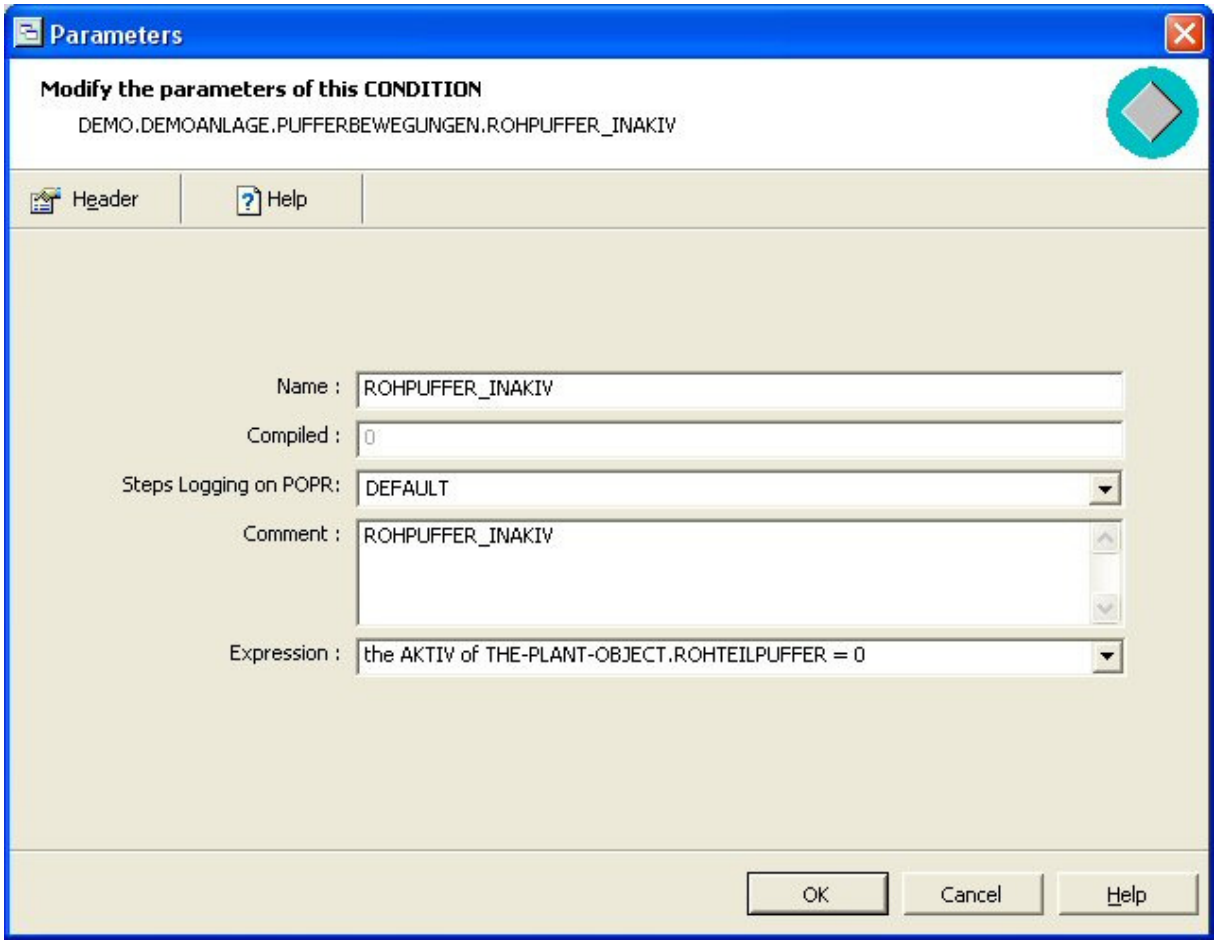

Abbildung 7.28: Parameterfenster eines „Condition-Objekts“

Für die Felder „Name“ und „Comment“ genügt es, wenn man nur in das Feld „Name“ eine Bezeichnung eingibt. Der Inhalt wird dann automatisch in das Feld „Comment“ übernommen.

Im Feld „Expression“ wird die eigentliche Bedingung erstellt. Die Bedingung im Beispiel „ROHPUFFER_INAKTIV“ soll überprüfen, ob das Attribut „AKTIV“ des Rohteilpuffers „FALSE“ (also „0“) ist. Folgende Eingaben sind zu tätigen:

- Name: ROHPUFFER_INAKTIV
- Comment: ROHPUFFER_INAKTIV
- Expression: the AKTIV of THE-PLANT-OBJECT.ROHTEILPUFFER=0

Um nicht die ganze „Expression“-Zeile eingeben zu müssen, kann man in der Drop-Down-Liste erst die Unit und das dazugehörige Attribut auswählen und muss dann nur mehr die mathematische Bedingung eingeben.

Für die anderen zwei „Condition“-Objekte sind hier nur mehr die jeweiligen „Expressions“ angeführt:

- HUT_IN_BES kontrolliert, ob eine Palette (Handling Unit) in der Be-/Entladestation ist.

  Expression: the BELADEN of THE-PLANT-OBJECT.B_E_STATION=1

- ROHPUFFER_LEER überprüft, ob keine Palette im Rohteilpuffer liegt.

  Expression: the BELADEN of THE-PLANT-OBJECT.ROHTEILPUFFER=0

- Method-Caller „BESZUPUFFER“: Dieser Method-Caller ruft, wie der „ZUSTANDPR“-Method-Caller, ebenfalls eine COM-Methode auf. Diese Com-Methode gibt dann den eigentlichen GO-Befehl zum Ausführen der Bewegung an die Anlage weiter. Die folgenden Eingaben sind im Parameterfenster durchzuführen:

  - Name: BESZUPUFFER
  - Comment: BESZUPUFFER
  - Owner-Class: the symbol DEMO.DEMOANLAGE.COM (Drop-Down-Liste)
  - Method-To-Call: BESZUPUFFER (Drop-Down-Liste)

  Dieser Method-Caller bleibt aktiv, so lange der Arbeitsschritt an der Anlage dauert. Die Regel geht also erst zum nächsten Objekt, wenn die Bewegung ausgeführt ist, d.h. die Palette von der Be-/Entladestation zum Rohteilpuffer transportiert wurde. Da die Palette im Material Manager aber noch immer in der Be-/Entladestation liegt, muss diese auch noch an den Rohteilpuffer übergeben werden. Dies geschieht mit zwei speziellen Method-Callern. Die GSI-System-Method-Caller.MM sind spezielle Method-Caller, denen Methoden für den Material Manager hinterlegt sind.

- GSI-System-Method-Caller.MM „FIND_HUT_IN_BES“: Um ein Handling Unit (also eine Palette) im Material Manager zu verschieben, muss man erst die genaue Identität der Palette kennen. Die Identität der Palette in der Be-/Entladestation wird durch diesen Method-Caller ermittelt. Folgende Eingaben im Parameterfenster sind durchzuführen:
  - Name: FIND_HUT_IN_BES
  - Comment: FIND_HUT_IN_BES
  - Method-To-Call: MMRunTime.FindHut

  Die Methode „FindHut“ hat nun wieder spezielle Argumente, die in der Registerkarte „Data“ angezeigt werden (Abbildung 7.29: Argumente der "FindHut"-Methode).

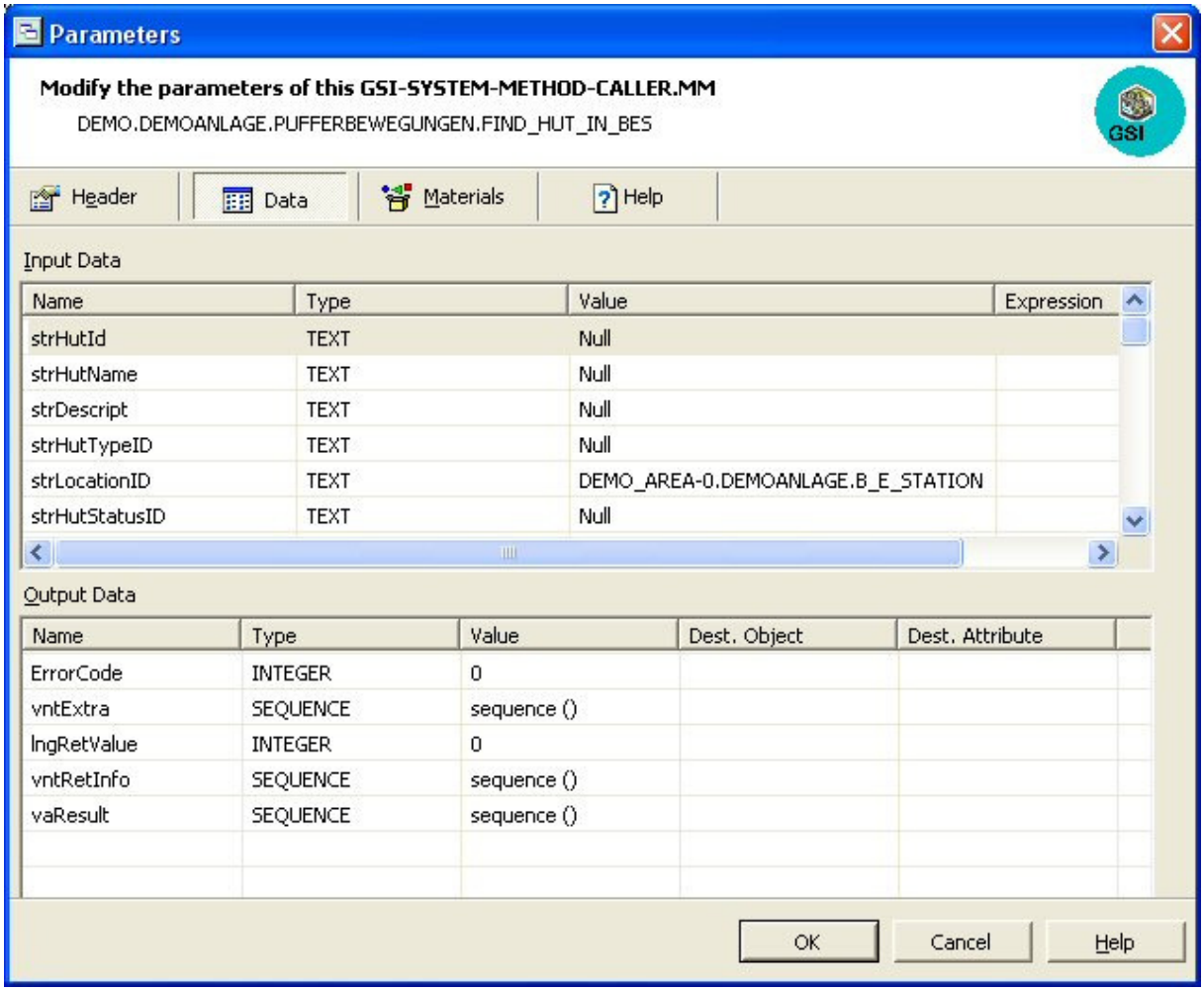

Abbildung 7.29: Argumente der "FindHut"-Methode

Diese Methode bietet verschiedene Möglichkeiten, um eine HUT zu finden. In diesem Beispiel sucht man aber die Palette, die an einer bestimmten Location liegt. Deshalb geben wir nur beim Argument „strLocationID“ Folgendes ein:

- Value: DEMO_AREA-0.DEMOANLAGE.B_E_STATION

Der obige Ausdruck wird im Feld „Value“ des Arguments „strLocationID“ eingegeben und bezeichnet den Ort Be-/Entladestation in der Zelle.

Beim Durchführen der Methode werden alle Informationen ermittelt, die die HUT in der BES betreffen und in das Outputargument „vaResult“ geschrieben. Dieses Argument hat den Type „Sequence“. Eine solche Sequence ist im Grunde genommen ein Array. Näher wird dieses Argument im nächsten Punkt beschrieben.

- Display-Object: Ein solches Objekt ist oft sehr hilfreich, da es Inhalte von Argumenten anzeigen kann. Es erleichtert das Ansprechen eines genauen Elementes einer Sequence. Man baut das Display-Object nach dem „FIND_HUT“-Method-Caller ein. Wenn man das Objekt in den Arbeitsbereich gezogen hat, öffnet sich gleich das Parameterfenster. Dieses ist gleich wieder zu schließen, da man zuerst das Objekt mit dem Regelstrang verbinden muss, um das gewünschte Argument anzeigen zu können. Hat man das Display-Object verbunden, öffnet man das Parameterfenster erneut, bestätigt mit dem „OK“-Button und kommt nun zum Fenster Expression. Hier wählt man in der Drop-Down-Liste das gewünschte Argument aus. In diesem Fall das „vaResult“-Argument des „FIND-HUT“-Method-Callers. Wird die Regel nun ausgeführt, sieht ein Display-Object wie in Abbildung 7.30 aus.

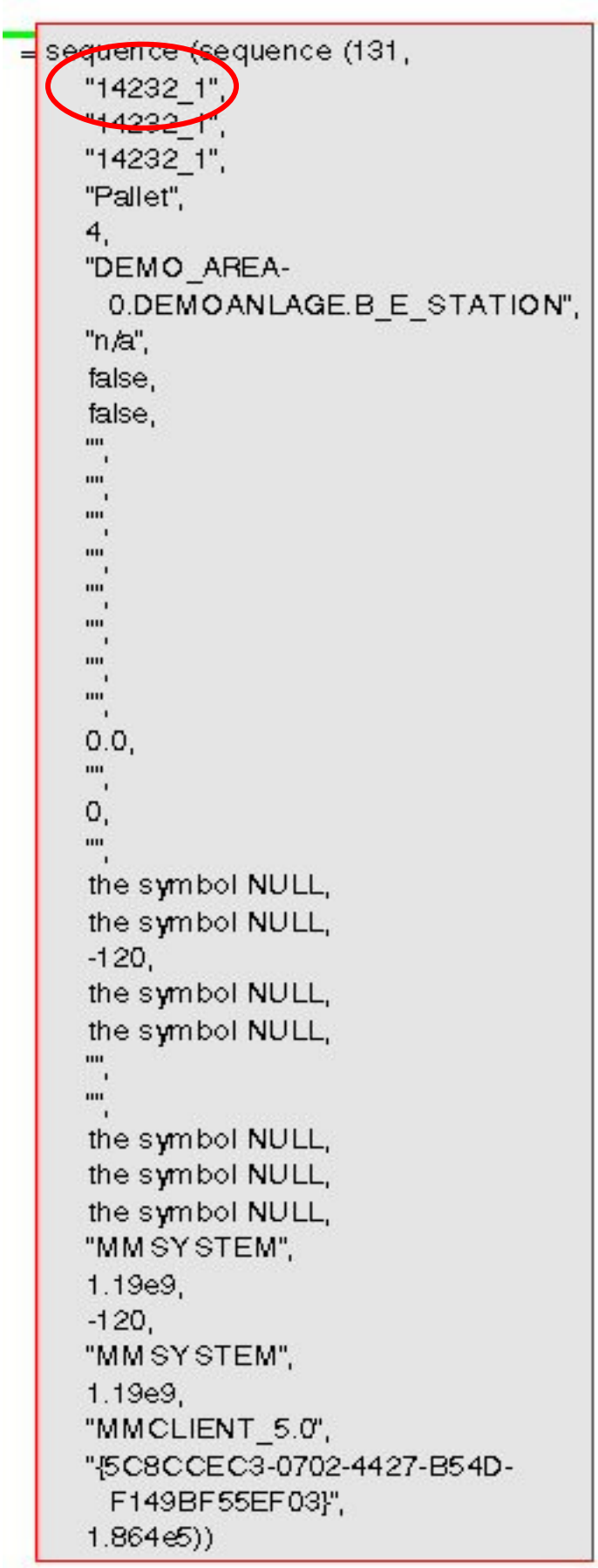

Abbildung 7.30: Inhalt eines Display-Objekts (Ergebnis einer „FindHut“Methode)

Der Inhalt von Abbildung 7.30 ist sozusagen das Ergebnis des „FIND-HUT“-Method-Callers. Der Inhalt besteht aus einer Haupt-Sequence und jede Palette am gesuchten Palettenplatz ist eine eigene Unter-Sequence. Normalerweise gibt es aber, wie im hier beschriebenen Beispiel, nur eine Unter-Sequence, da an einem Palettenplatz nur eine Palette liegen kann. In dieser Unter-Sequence ist eine ganze Reihe von Informationen enthalten. Man will aber nur die Identität der gefragten Handling Unit herausfinden. Diese ID ist in der dargestellten Sequence im rot gekennzeichneten Bereich abzulesen. Wie beim Herausfiltern der ID vorgegangen werden muss, ist bei der Erklärung des „MOVEHUT_ZU_ROHPUFFER“-Method-Callers beschrieben.

- Method-Caller „MOVEHUT_ZU_ROHPUFFER“: Dieser Method-Caller transportiert nun die im MM auf der BES liegende Palette in den Rohteilpuffer.
  - Method-To-Call: MMRunTime.DoMoveHut

  Es werden in der Registerkarte „Data“ wieder methodenspezifische Argumente angezeigt, wobei nur die ID der zu transportierenden HUT (strHutId) und der Zielort (strTargetLocPath) wichtig ist.

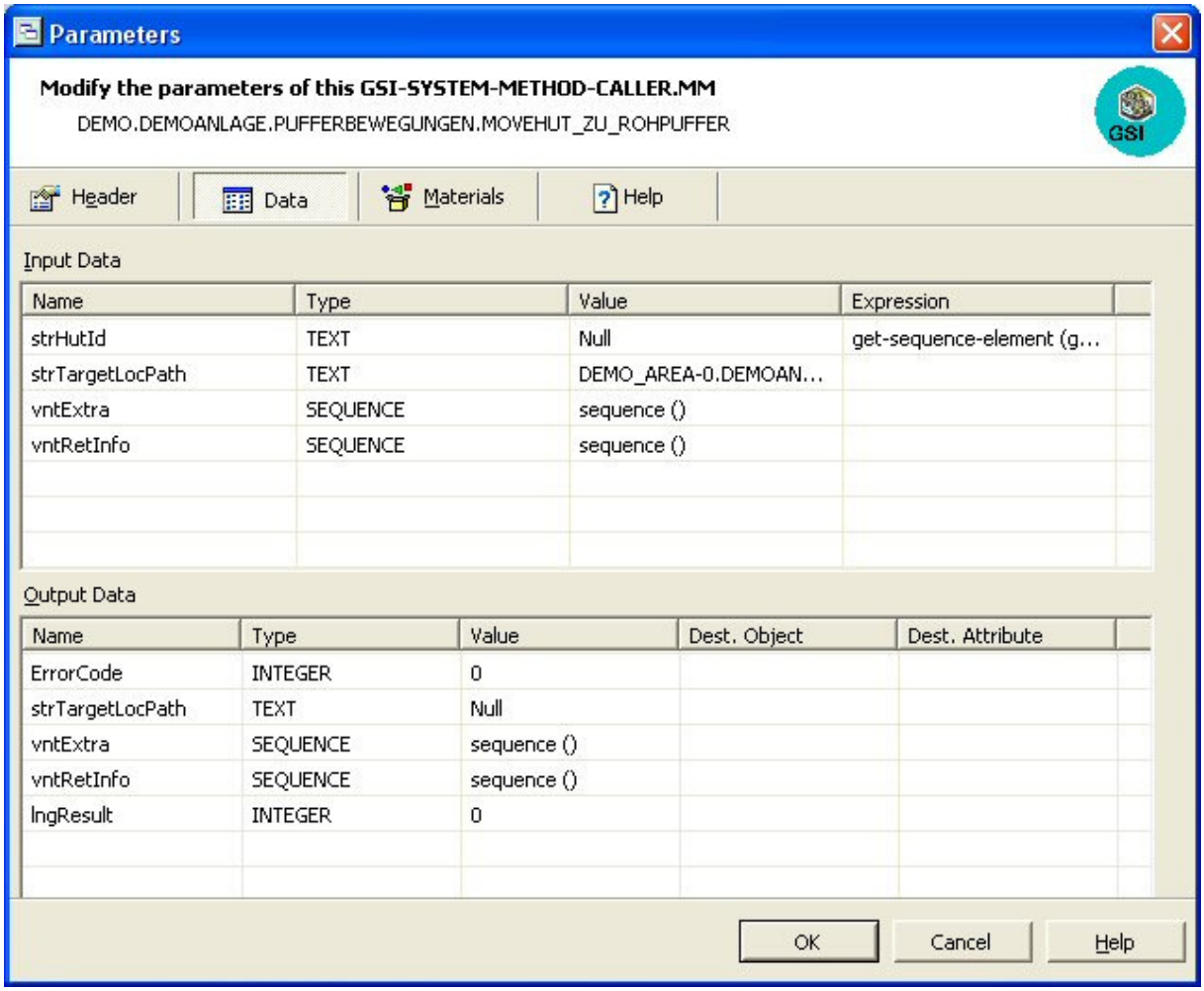

Abbildung 7.31: Argumente der „MoveHut“-Methode

„strHutId“: Dieses Argument gibt die Identität der zu verschiebenden Palette an. Man kann aber nicht einfach ein Value einsetzen, da die ID ja palettenabhängig ist und deshalb wechselt. Aus diesem Grund wurden ja im vorigen Method-Caller alle Daten der in der BES liegenden Palette bestimmt. Diese Eigenschaften sind in Form einer Sequence-Variable im Argument „vaResult“ des „FIND_HUT_IN_BES“ abgelegt. Die Aufgabe des Users ist es nun, die „HutId“ aus dieser Sequence-Variable herauszufiltern. Dies geschieht im Feld Expression des Arguments „strHutId“.

Da die Syntax für den Befehl etwas komplexer ist, wird dies nun genauer beschrieben. Zuerst muss also das Argument „vaResult“ des „FIND-HUT“-Method-Callers aufgerufen werden. Dies geschieht mit

```
get-argument-
value(DEMO.DEMOANLAGE.PUFFERBEWEGUNGEN.FIND_HUT_IN_BES, the symbol
RETURN, "vaResult"), the symbol RETURN, "vaResult".
```

Man hat also die gesamte Sequence und als Nächstes soll man aus der Haupt-Sequence die erste Unter-Sequence mit den Palettendaten auswählen. Dazu benutzt man den ersten „get-sequence-element“-Befehl, der wie folgt aussieht: „get-sequence-element(Sequence, Elementnummer)“. Die Elementnummer beginnt bei null zu zählen. Um also das erste Element anzusprechen, gibt man „0“ ein. In unserem Beispiel sieht das wie folgt aus:

```
get-sequence-element (get-argument-
value(DEMO.DEMOANLAGE.PUFFERBEWEGUNGEN.FIND_HUT_IN_BES, the symbol
RETURN, "vaResult"),0).
```

Jetzt hat man die Sequence der gesuchten Handling Unit und will nun noch die ID auswählen. Die ID ist er zweite Eintrag in der HUT-Sequence (genaue Beschreibung der einzelnen Einträge findet man in der COM- und GSI- Schnittstellenhilfe für Komponenten der SIMATIC IT Production Suite) und wird mit einem weiteren „get-sequence-elements“-Befehl bestimmt. Der gesamte Ausdruck lautet dann wie unten beschrieben.

Expression: get-sequence-element (get-sequence-element (get-argument-value(DEMO.DEMOANLAGE.PUFFERBEWEGUNGEN.FIND_HUT_IN_BES, the symbol RETURN, "vaResult"),0),1)

„strTargetLocPath“: Der Zielort für die Regel „BES_ZU_PUFFER“ ist mit dem Rohteilpuffer eindeutig bestimmt und kann somit ins Feld Value eingetragen werden.

Value: `DEMO_AREA-0.DEMOANLAGE.ROHTEILPUFFER`

- End-Of-Rule: Ganz am Ende der Regel wird noch ein End-Of-Rule-Objekt eingebaut. Dieses ist nicht bei jeder Regel erforderlich. In diesem Fall allerdings schon, da es ein Wait-For-Alarm-Objekt gibt. Dieses Objekt würde kein automatisches Ende der Regel möglich machen. Deshalb muss die Regel mit einem

End-Of-Rule-Objekt beendet werden. Bei diesem Objekt wird im Parameterfenster der „Exit-Status“ auf „OK“ gesetzt, um ein positives Ende der Regel zu signalisieren.

Die Regel ist nun so weit fertig, um ausgeführt werden zu können. Falls es aber zu einem Fehler kommt, würde die Regel einfach beendet, ohne dass der Benutzer eine Rückmeldung bekommt. Aus diesem Grund wird jetzt noch festgelegt, was geschehen soll, wenn ein Fehler Auftritt.

Tritt bei den einzelnen „Conditions“ ein Fehler auf, werden über Method-Caller Fehlermeldungen aufgerufen. Hier gibt es zwei verschiedene Möglichkeiten. Diese sind eine Fehlermeldung, falls das System aktiv ist, oder eine Fehlermeldung, wenn ein Problem bei den Palettenplätzen auftritt.

Tritt während der Regeldurchführung ein Fehler in der Anlage auf, schickt diese einen Alarm an das MES. Dieser Alarm wird, solange die Regel aktiv ist, von einem „Wait-For-Alarm“-Objekt abgefragt. Ist der Alarm-Event ausgelöst, wird ebenfalls eine Fehlermeldung am Bildschirm ausgegeben.

- Wait-For-Alarm: Wird dieses Objekt aufgerufen, bleibt es solange aktiv, bis der hinterlegte Event eintritt. Erst dann läuft die Regel in diesem Strang weiter ab. Hier ist der Event ein Alarm der Anlage. Dieser Event ist wiederum im gleichen COM-Objekt hinterlegt wie zum Beispiel die Methode „ZUSTANDPR“. Neben „Name“ und „Comment“ sind im Parameterfenster noch die unten angeführten Einstellungen zu treffen.

  Owner-Class: `the symbol DEMO.COM`

  Event-to-Wait-for: `the symbol ALARM`

- Message-Method-Caller: Diese sind ganz normale Method-Caller, die eine Methode aufrufen. Diese Methode ruft dann wiederum eine Regel auf, die dann eine Fehlermeldung generiert. Eine solche Vorgehensweise ist von Vorteil, wenn man dieselbe Regel von mehreren Orten aus aufrufen möchte. Man braucht die Regel dann nur einmal programmieren und ruft sie mit einem Method-Caller auf. Diese Methode muss aber vorher erst definiert werden.

  Method-Caller „SYSTEM_AKTIV“:

  - Owner-Class: `the symbol DEMO.DEMOANLAGE`
  - Method-to-Call: `the symbol SYSTEM_AKTIV`

  Method-Caller „WARNUNG PALETTE“:

  - Owner-Class: `the symbol DEMO.DEMOANLAGE`
  - Method-to-Call: `the symbol WARNUNG_PALETTE`

- Method-Caller „ALARM“: Bei diesem Method-Caller gibt es noch die Besonderheit, dass an die Methode noch der Regelname mitübergeben wird. Dazu wird neben den normalen Eingaben in der Registerkarte „Data“ noch ein Inputargument erzeugt. Dieses Argument hat den Typ TEXT und im „Value“ steht der Regelname (BES_ZU_PUFFER).
  - Owner-Class: `the symbol DEMO.DEMOANLAGE`
  - Method-to-Call: `the symbol ALARM`

- End-of-Rule: Am Ende dieses Teils wird dann noch ein End-Of-Rule-Objekt eingefügt und der „Exit-Status“ auf „NOT-OK“ gesetzt. Dies verdeutlicht, dass die Regel nicht korrekt ausgeführt wurde.

7.4.4 Kurzbeschreibung aller Regeln

7.4.4.1 Auftragsmanagement

Im Regel-Container Auftragsmanagement werden feingeplante Aufträge aus dem ERP ins MES übernommen und in den POM eingetragen. Dies geschieht über das so genannte Data Integration Service (DIS). Es muss zuerst eine Verbindung durch die „Connection“-Regel über einen DIS-GSI-Method-Caller zum DIS-Server aufgebaut werden.

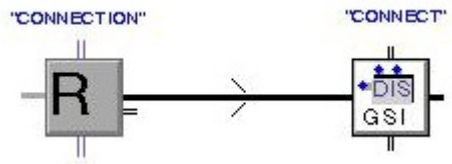

Abbildung 7.32: Connection-Regel

Nachdem die Verbindung steht, kann die Auftragsregel (Abbildung 7.33) gestartet werden. Diese Regel beinhaltet eine Endlosschleife. In der Schleife wartet dann ein DIS-GSI-System-Wait-for-Event auf eine Nachricht vom DIS-Server. Diese Nachricht wird geschickt, wenn ein neuer Auftrag über einen Connector eingelesen wurde. Ein GSI-System-Method-Caller.dis bestätigt dann den Nachrichteneingang. Ein weiterer solcher Method-Caller ruft die Nachricht ab und ein System-Method-Caller.pom liefert die Daten an den Production Order Manager Server.

Abbildung 7.33: Auftragsregel

Hat ein Auftrag benutzerdefinierte Inhalte (Custom Fields) wie zum Beispiel ein NC-Programm, können diese mit folgender Regel beschrieben werden. Hierzu müssen aber die gewünschten Informationen über Argumente im ROOT-Objekt übergeben werden.

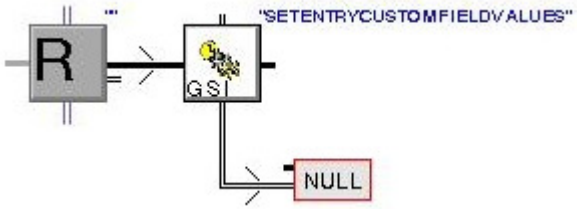

Abbildung 7.34: AG-Benutzerfelder setzen

7.4.4.2 Automatikbetrieb

Im Automatikbetrieb gibt es für jeden auszuführenden Prozess eine Regel. Bis auf die Regel „PAL_BESTUECKEN_AUT“ (in Abbildung 7.35 rot markiert) müssen alle Regeln von Anfang an aktiviert sein, da sie mit „Wait-for-Event“-Objekten arbeiten und somit ständig auf Signale der Anlage warten müssen.

Um eine Palette automatisch die gesamte Zelle durchlaufen zu lassen, muss die Regel „PAL_BESTUECKEN_AUT“ manuell über die CAB-Bedieneroberfläche aktiviert werden. Von dieser Regel wird dann die Methode „PAL_BESTUECKEN“ aufgerufen. Ist der Vorgang abgeschlossen, sendet die Regel ein Signal (Event), um den Vorgang fertig zu melden. Das „Wait-for-Event“-Objekt der „BES_ZU_PUFFER_AUT“-Regel wartet nun schon auf dieses Signal. Wird der Event ausgelöst, startet die zur Regel gehörende Methode. Nach deren Durchführung wird wiederum ein Event zur Fertigmeldung gesendet und die nachfolgende Regel kann durchlaufen werden. So durchläuft die Palette die ganze Anlage, bis sie wieder bei der Be-/Entladestation eingelangt ist. In der folgenden Abbildung ist mit roten Pfeilen gekennzeichnet, welcher „Event-Sender“ welchen „Wait-for-Event“ aufruft.

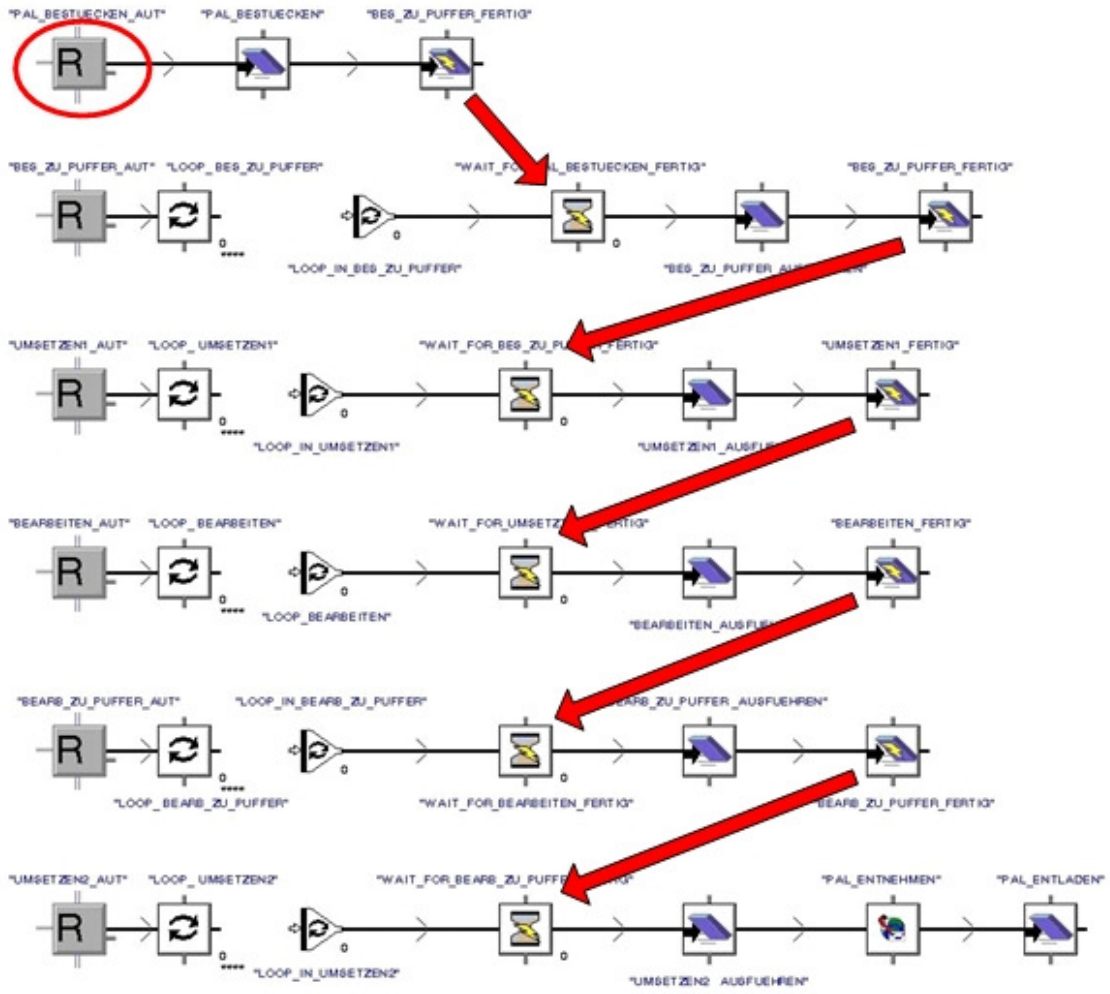

Abbildung 7.35: Regeln im Automatikbetrieb

7.4.4.3 Bearbeitung

In der Bearbeitungsregel wird für jedes Stück eine Schleife durchlaufen. In der Schleife wird ein Sublost im MM erstellt und das Startkommando an den Roboter gesendet. Die Regel sieht wie folgt aus:

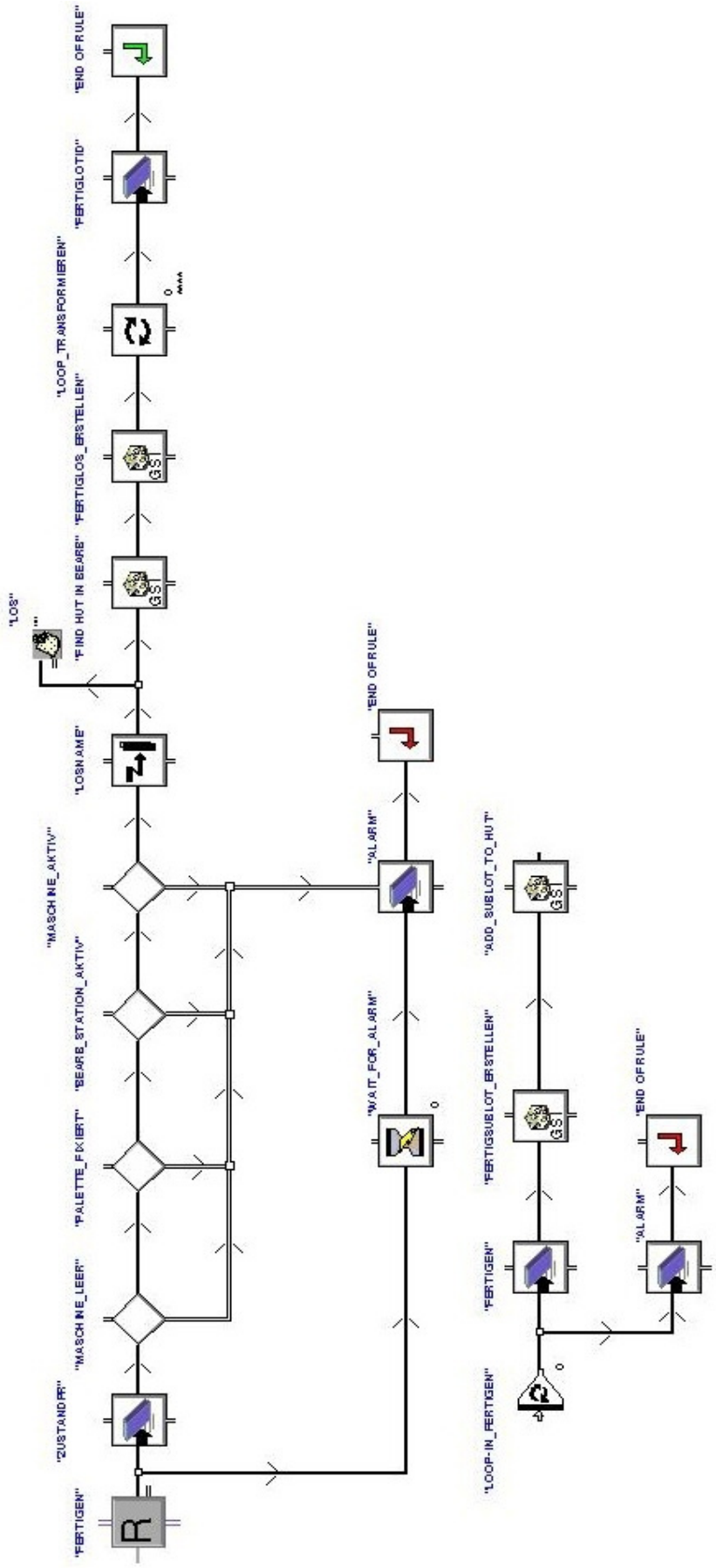

Abbildung 7.36: Bearbeitungsregel

7.4.4.4 Initialisierung

Die Initialisierung soll die Anlage (Simulator) in den Ausgangszustand bringen. Zusätzlich werden alle Regeln gestartet, die im Hintergrund mitlaufen müssen. Dies sind die Regeln für den Automatikbetrieb und für das Auftragsmanagement. Die Regel wird über die CAB-Bedieneroberfläche aufgerufen.

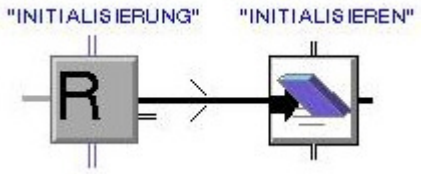

Abbildung 7.37: Initialisierungsregel

7.4.4.5 Nachrichtenmanagement

Im Nachrichtenmanagement sind Regeln hinterlegt, die Nachrichten für den Benutzer generieren. Sie werden von verschiedenen Regeln über Method-Caller aufgerufen.

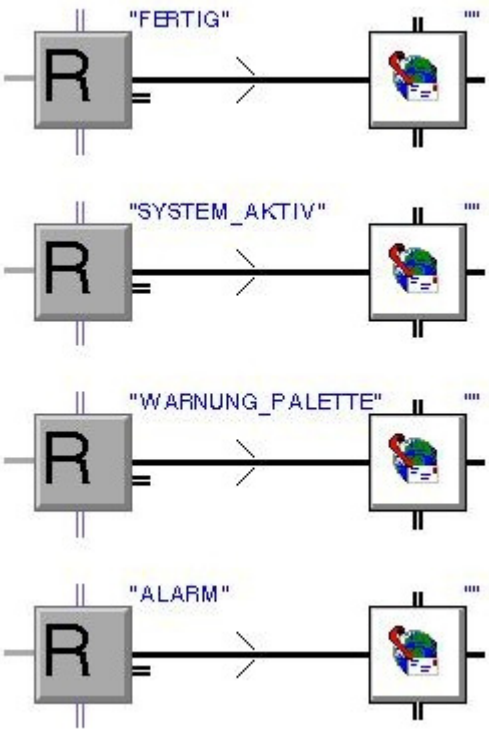

Abbildung 7.38: Regeln im Nachrichtenmanagement

7.4.4.6 Pufferbewegungen

Die Pufferbewegungen werden durch sechs Regeln dargestellt. Diese sind im folgenden Teil kurz beschrieben.

- Palette einlegen

In dieser Regel wird zuerst abgefragt, für welche Teile die Palette gedacht ist, nach der Anzahl der Teile, die darauf abgelegt werden sollen und eine Paletten-ID. Anschließend wird abgefragt, ob die Palette groß genug für das Los ist und ob die Be-/Entladestation frei ist. Anschließend wird dann die HUT über einen „GSI-System-Method-Caller.mm“ im Material Manager hinterlegt.

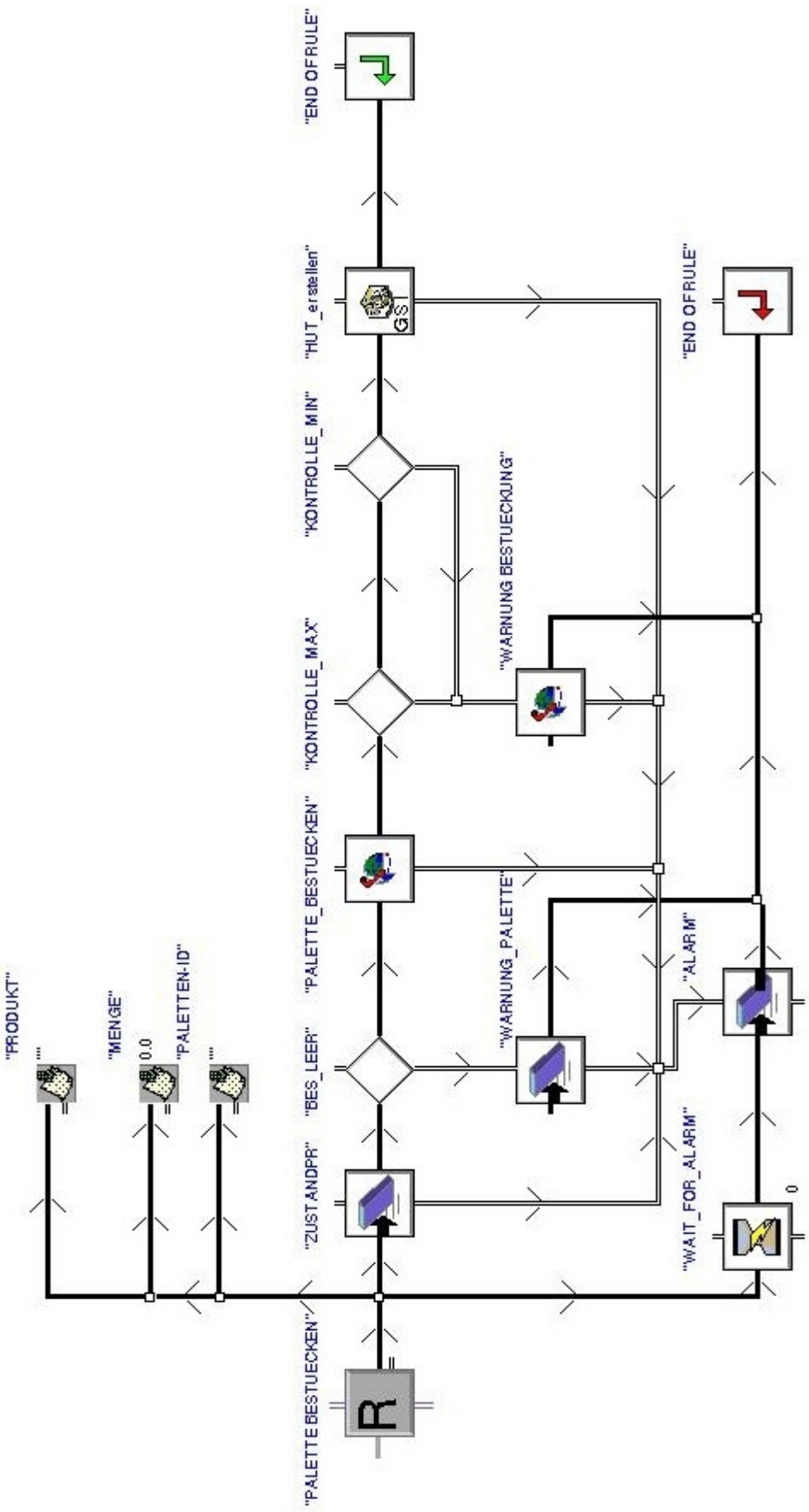

Abbildung 7.39: Palette einlegen

- Die vier Transportregeln (BES_ZU_PUFFER, UMSETZEN_1, UMSETZEN_2, BEARB_ZU_PUFFER)

Diese Regel gibt den GO-Befehl für den Transport an die Anlage weiter. Zuerst wird aber mit Condition-Elementen abgefragt, ob die Bewegung gerade ausgeführt werden kann. Neben dem GO-Befehl muss aber auch noch die Palette im Material Manager an den richtigen Ort übergeben werden. Dies dient zur Nachverfolgung der Lose.

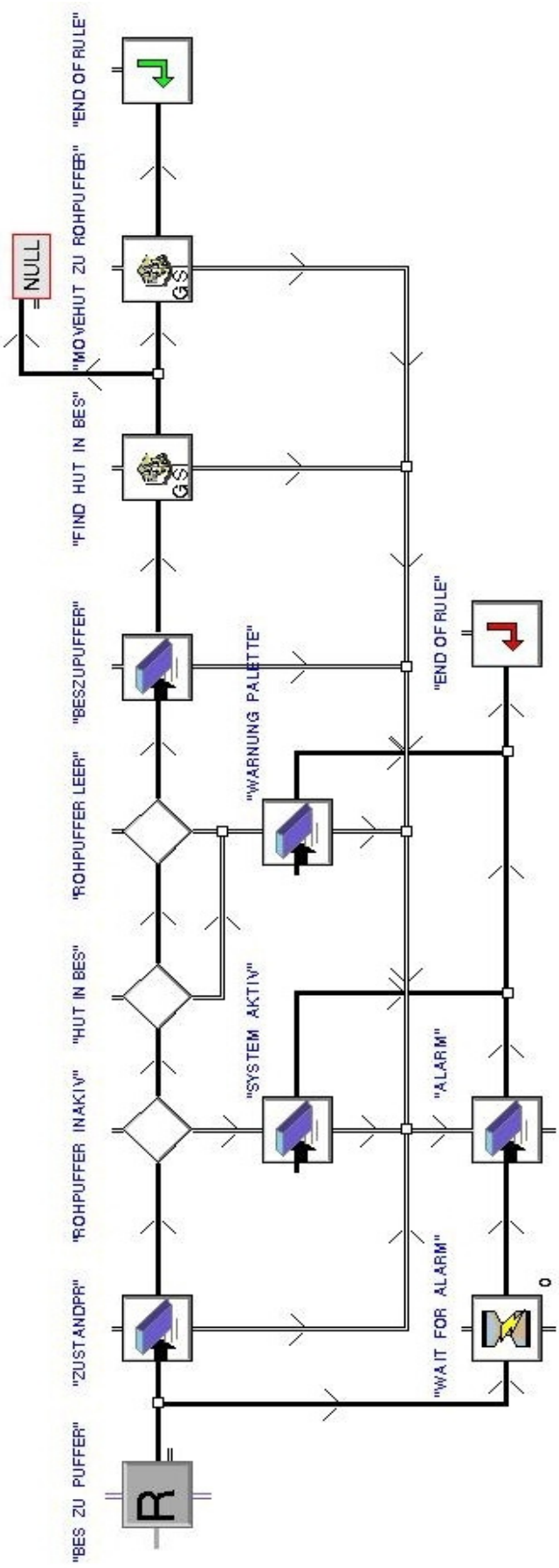

Abbildung 7.40 BES_ZU_PUFFER

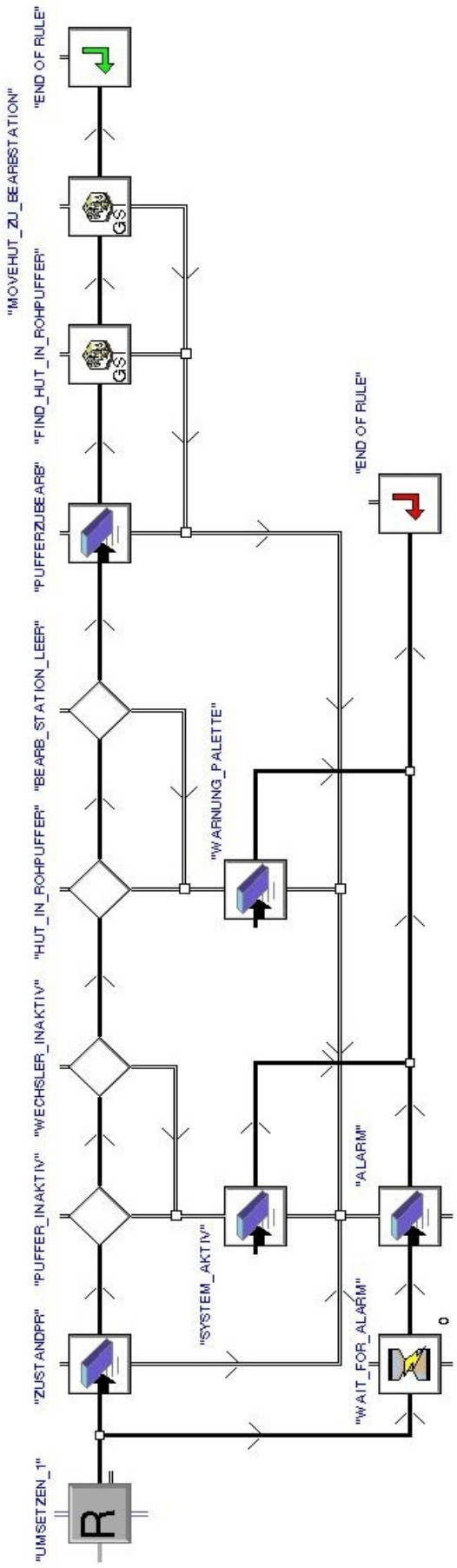

Abbildung 7.41: UMSETZEN_1

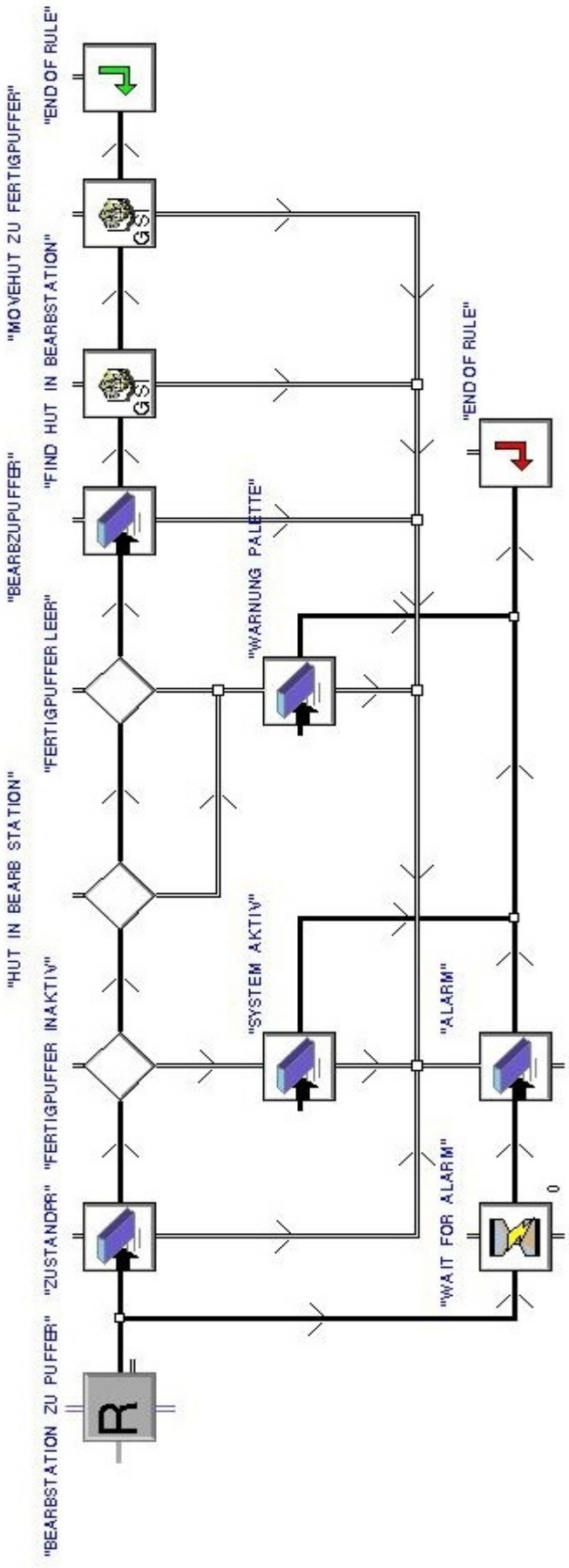

Abbildung 7.42: BEARBEITUNGSSTATION_ZU_PUFFER

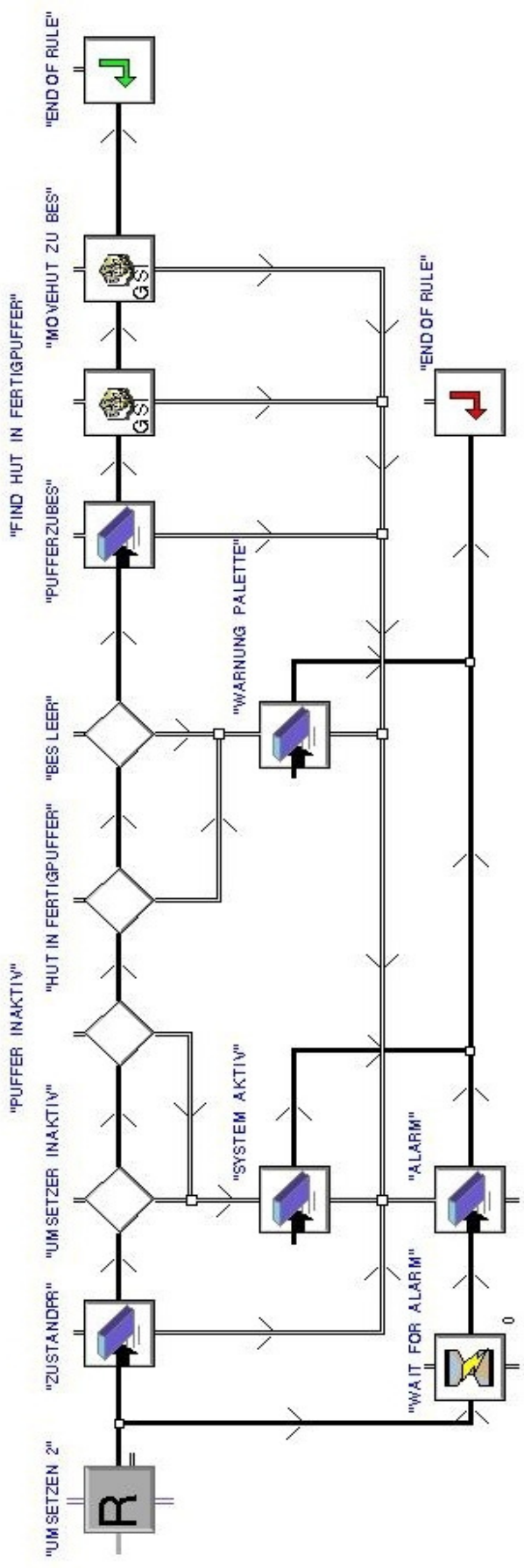

Abbildung 7.43: UMSETZEN_2

- Palette entnehmen

Hier werden zuerst in einer Schleife alle vorhandenen Sublose gelöscht und in einer zweiten Schleife dann die Lose. Ist kein Sublos oder Los vorhanden, wird bei den GSI-Method-Callern „SUBLOTS_IN_HUT“ und „LOTS_IN_HUT“ der negative Ausgang verwendet. Zum Schluss wird noch die HUT, also die Palette, gelöscht.

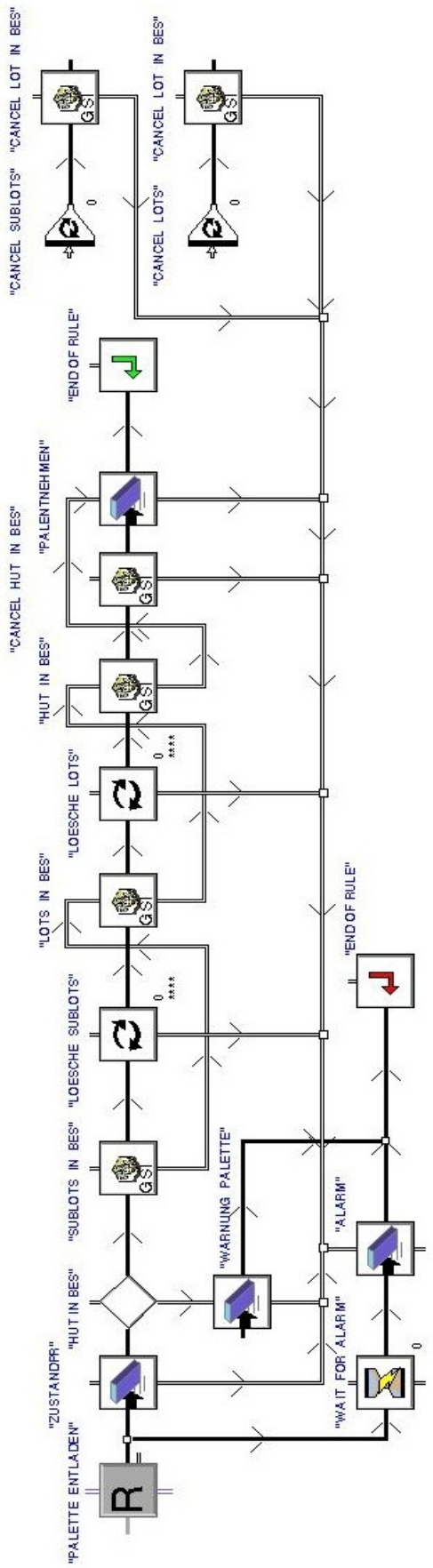

Abbildung 7.44: PALETTE_ENTLADEN

7.5 Anlage im Arbeitsbereich erstellen

7.5.1 Allgemeines

Bisher wurden Bibliotheken mit Anlagenkomponenten modelliert. Diese sind aber noch nicht mit physikalischen Objekten verbunden. Um diese Verbindung herzustellen, muss erst ein neuer Standort im Arbeitsbereich erstellt werden. Wenn dieser Standort erstellt ist, kann er mit Komponenten aus der Bibliothek gefüllt werden. Zum Schluss muss die Anlage noch instanziert werden.

7.5.2 Vorgehensweise

Man navigiert im Arbeitsbereich des Production Modelers in der Baumstruktur auf „PLANT-DEFINITION“. Danach drückt man die Schaltfläche „Hinzufügen von Elementen“.

Statt der Baumstruktur sieht man nun die einzelnen Bibliotheken aufgelistet. Man wählt nun die Bibliothek „DEMO“ aus, danach erscheint die „Demo_Area“. Diese wählt man aus und zieht sie in das rechte Fenster. Nun wird noch das Dialogfeld „Parameters“ angezeigt. Hier kann man noch einen Namen eingeben und auf „OK“ klicken.

Die Anlage ist somit instanziert und kann in Betrieb gehen.

Werden im Bibliotheksbereich nachträglich Änderungen vorgenommen, werden diese im Arbeitsbereich nicht automatisch aktualisiert. Aus diesem Grund gibt es im Production Modeler die Funktion „Synchronisieren“. Das Synchronisieren ist hier nicht näher beschrieben, da die von SIEMENS für SIMATIC IT Production Suite zur Verfügung gestellten Hilfe-Dateien ausreichend sind.

8 Bedienung verwendeter SIMATIC IT Komponenten

8.1 Bedienung des Production Order Managers (POM)

Der Production Order Manager (POM) ist jene Komponente der SIMATIC IT Production Suite, die für die Auftragsverwaltung und Planung zuständig ist.

In diesem Projekt läuft nur der POM-Server, da keine Aufträge über die POM-Bedieneroberfläche direkt eingegeben werden. Die Aufträge werden vom Feinplanungstool (Scheduler) im XML-Format in einem bestimmten Ordner abgelegt. Dort werden sie vom Production Modeler übernommen und im POM eingetragen. Dem POM obliegt auch das Statusmanagement der einzelnen Aufträge und Arbeitsgänge. Die Statuswechsel werden aber auch über den Production Modeler bzw. über die CAB-Bedieneroberfläche vorgenommen.

Die Bedieneroberfläche des Production Order Manager muss also während des Produktionsvorganges nicht geöffnet werden. Sie kann aber benutzt werden, um Informationen über einen Auftrag (Order) oder Arbeitsgang (Entry) abzurufen oder spezielle Änderungen vorzunehmen.

Im folgenden Teil sind die allgemeine Handhabung des POM, das Statusmanagement und das XML-Import-File beschrieben.

8.1.1 Allgemeine Handhabung des Production Order Manager

Um in das Hauptfenster der POM-Benutzeroberfläche zu gelangen, öffnet man in der Menüleiste der Managementkonsole unter Components – Production Order Manager den Presentation Client. Es öffnet sich nun ein leeres Fenster. Damit die Auftragsansicht angezeigt wird klickt man unter File – Open auf Orders. Das folgende Fenster wird nun angezeigt, siehe Abbildung 8.1.

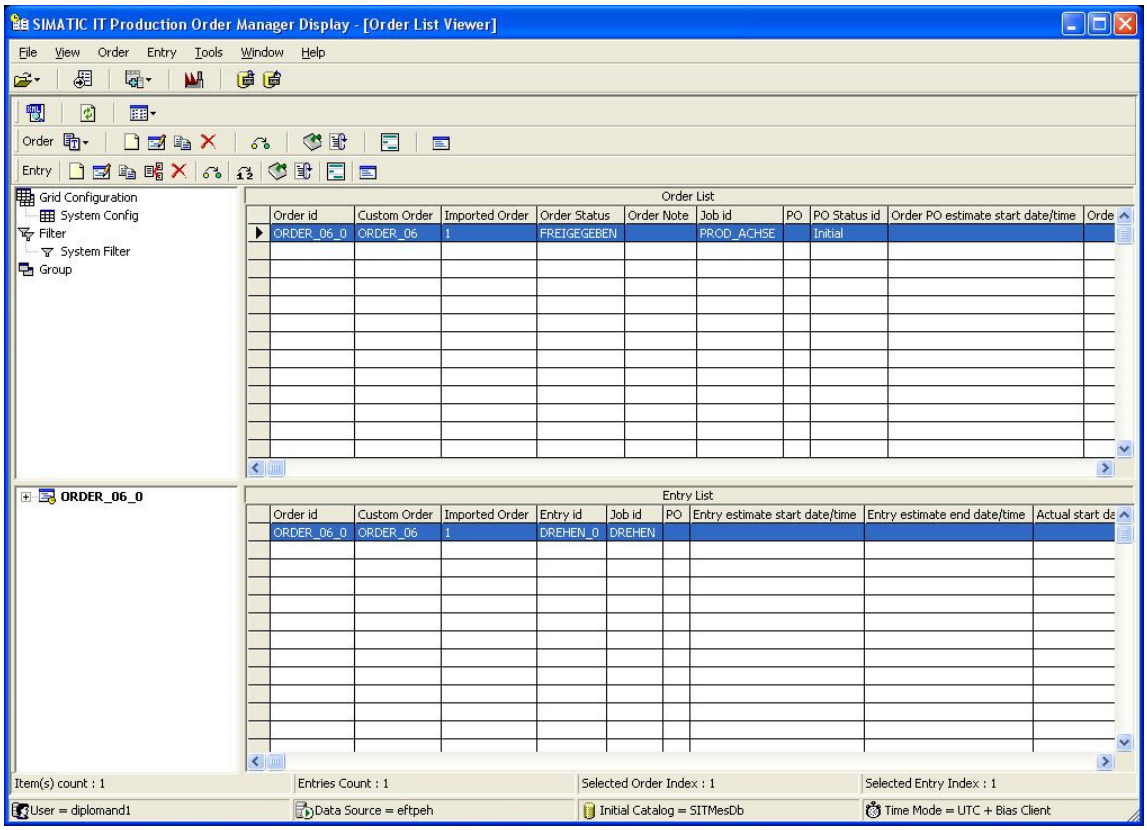

Abbildung 8.1: Production Order Manager Display

In der oberen rechten Tabelle (Order List) befinden sich alle Aufträge. Wählt man nun einen Auftrag aus, werden in der unteren Tabelle (Entry List) die dazugehörigen Arbeitsgänge angezeigt. Mit einem Rechtsklick auf einen Order oder Entry kann man sich die Details (Details) ansehen oder die Inhalte (Edit) bearbeiten. Ein Detail-Fenster ist in Abbildung 8.2 zu sehen.

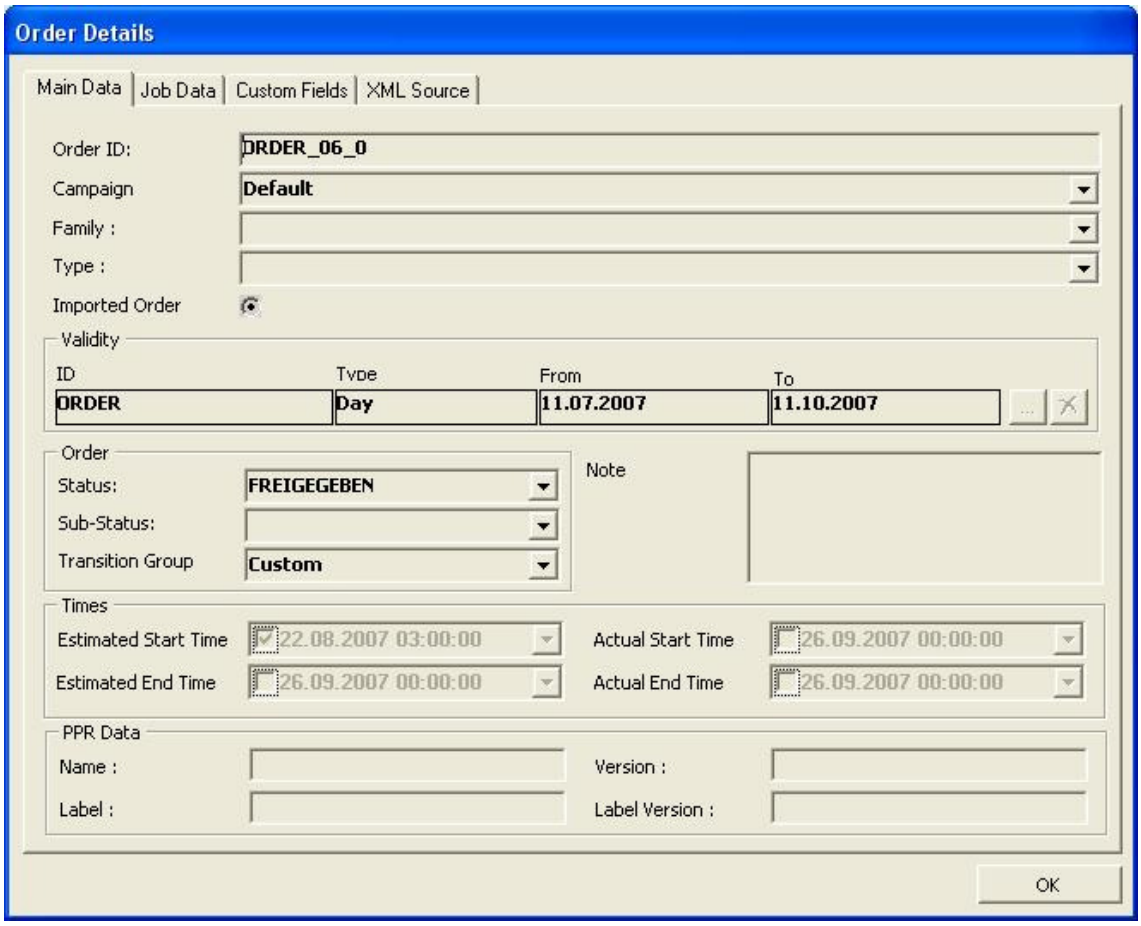

Abbildung 8.2: Order-Details

8.1.2 Status Management

Jeder Auftrag und jeder Arbeitsgang hat einen Status. Dieser Status gibt jeweils den aktuellen Zustand eines Auftrages oder Arbeitsganges an. Die folgenden beiden Abbildungen (Abbildung 8.3 und Abbildung 8.4) zeigen, welche möglichen Zustände ein Auftrag bzw. ein Arbeitsgang haben kann. In diesen Abbildungen ist auch ersichtlich, welche Statuswechsel möglich sind. Jeder Statuswechsel wird auch in der SIMATIC IT-Datenbank mitprotokolliert. Dies ermöglicht eine anschließende Auswertung des Auftrages.

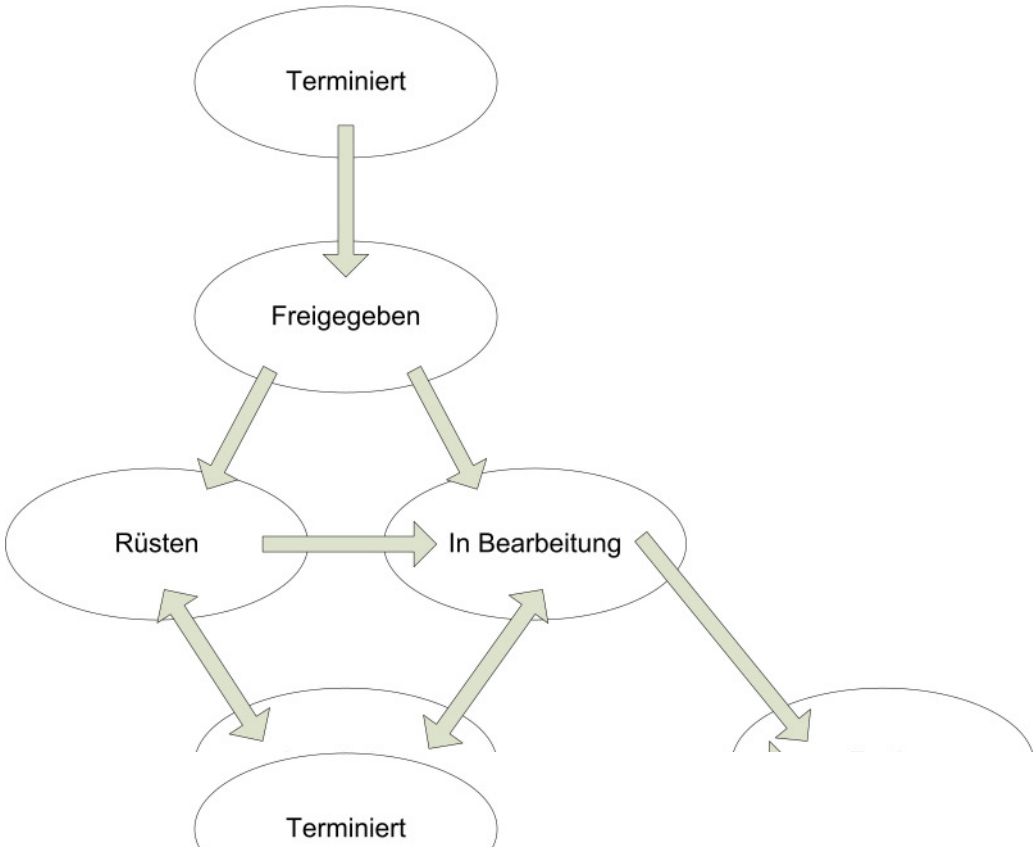

Abbildung 8.3: Statusübergänge bei Aufträgen

Bei einem Auftrag werden für dieses Projekt sechs verschiedene Zustände definiert. Die einzelnen Status sind wie folgt definiert:

- Terminiert: Der Auftrag hat diesen Status, wenn er vom Feinplanungstool zeitlich eingeplant worden ist.
- Freigegeben: Wird ein Auftrag ins MES übernommen, bekommt er automatisch den Status „Freigegeben“. Dies bedeutet, dass ab sofort mit der Bearbeitung oder dem Rüsten begonnen werden kann.
- Rüsten: Dieser Status wird gesetzt, wenn für ein Arbeitsgang gerade gerüstet wird.
- In Bearbeitung: Ist ein Arbeitsgang in Bearbeitung, wird auch der Auftragsstatus auf „In Bearbeitung“ gesetzt.
- Angehalten: Bei jeder Unterbrechung wird dieser Status gesetzt.
- Fertig: Der Auftragsstatus wird auf „Fertig“ gesetzt, wenn alle einzelnen Arbeitsgänge abgeschlossen sind.

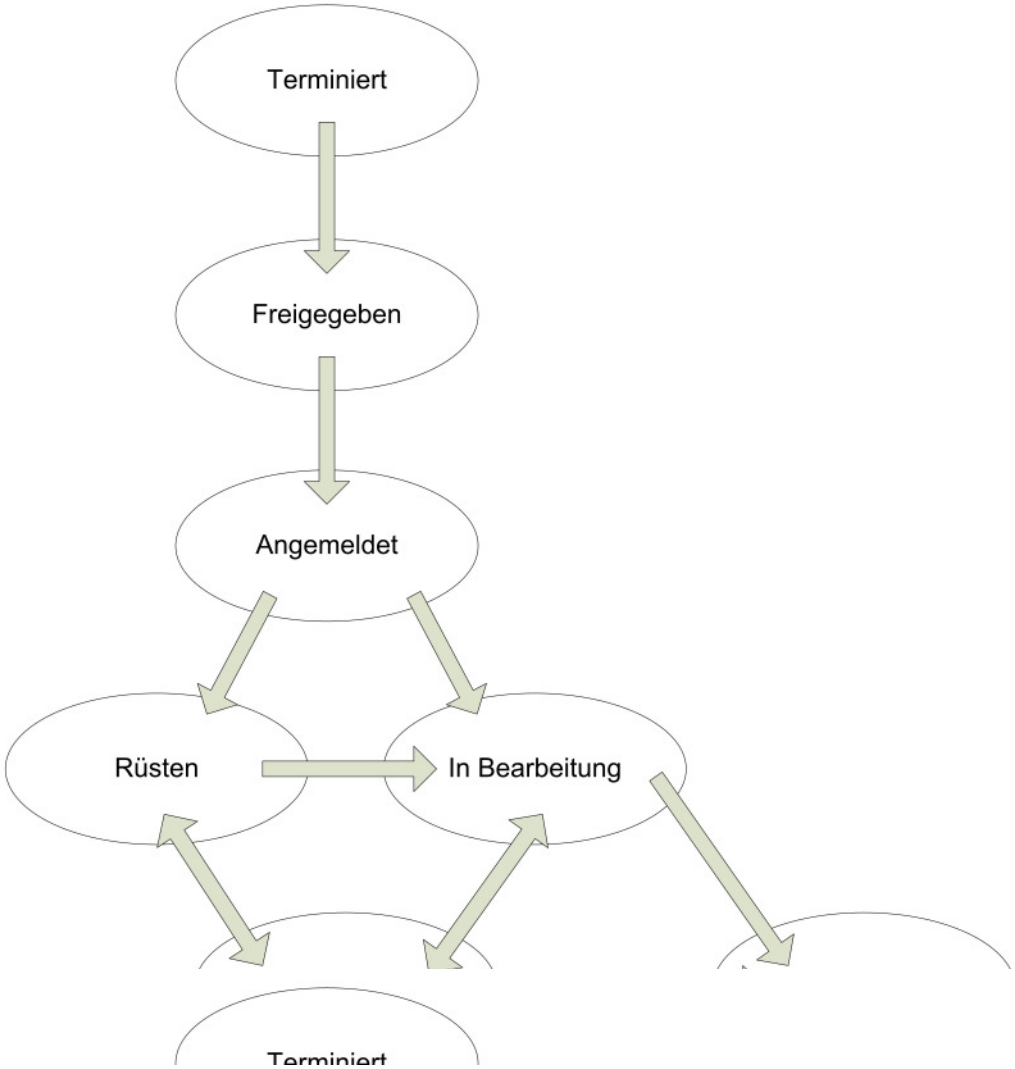

Abbildung 8.4: Statusübergänge bei Arbeitsgängen

Bei einem Arbeitsgang sind die einzelnen Zustände wie folgt definiert:

- Terminiert: Ist der Auftrag terminiert, sind auch die einzelnen Arbeitsgänge terminlich eingeplant.
- Freigegeben: Wenn der Auftrag freigegeben ist, wird auch automatisch der erste Arbeitsgang freigegeben. Ist dieser Arbeitsgang fertig, wird der nächste Arbeitsgang auf „Freigegeben“ gesetzt.
- Angemeldet: Dieser Status dient speziell der Handhabung eines Arbeitsganges in der Fertigungszelle dieses Projekts. Der Status wird vom Bediener der Fertigungszelle gesetzt, wenn ein Arbeitsgang begonnen wird.
- Rüsten: Ist ein Rüsten für den Arbeitsgang notwendig, wird der Status auf „Rüsten“ gesetzt.
- In Bearbeitung: Wenn sich eine Palette in der Bearbeitungsstation befindet, und wenn gerade Werkstücke bearbeitet werden, wird dieser Status gesetzt.

- Angehalten: Ein angemeldeter Arbeitsgang wird auf „Angehalten“ gesetzt, wenn er nicht gerade gerüstet oder bearbeitet wird.
- Fertig: Sind in der Fertigungszelle alle Paletten eines Arbeitsganges entnommen, wird der Auftrag vom Bediener auf Fertig gesetzt und der Folgearbeitsgang wird somit freigegeben.

8.1.2.1 Erstellen von Zuständen und dazugehörigen Übergangsgruppen

Es gibt im POM vordefinierte Status und dazugehörige Übergangsgruppen (so genannte Transition Groups). Da die angegebenen Zustände verwendet werden sollen, muss man diese erst definieren, ebenso wie mögliche Statuswechsel. Die Vorgangsweise ist bei Aufträgen und Arbeitsgängen gleich. Es ist deshalb nur die Vorgehensweise bei den Aufträgen genauer erläutert.

Im POM ist in der Menüleiste unter Tools das Statusmanagement auszuwählen. Das Fenster Statusmanagement sieht wie folgt aus (Abbildung 8.5):

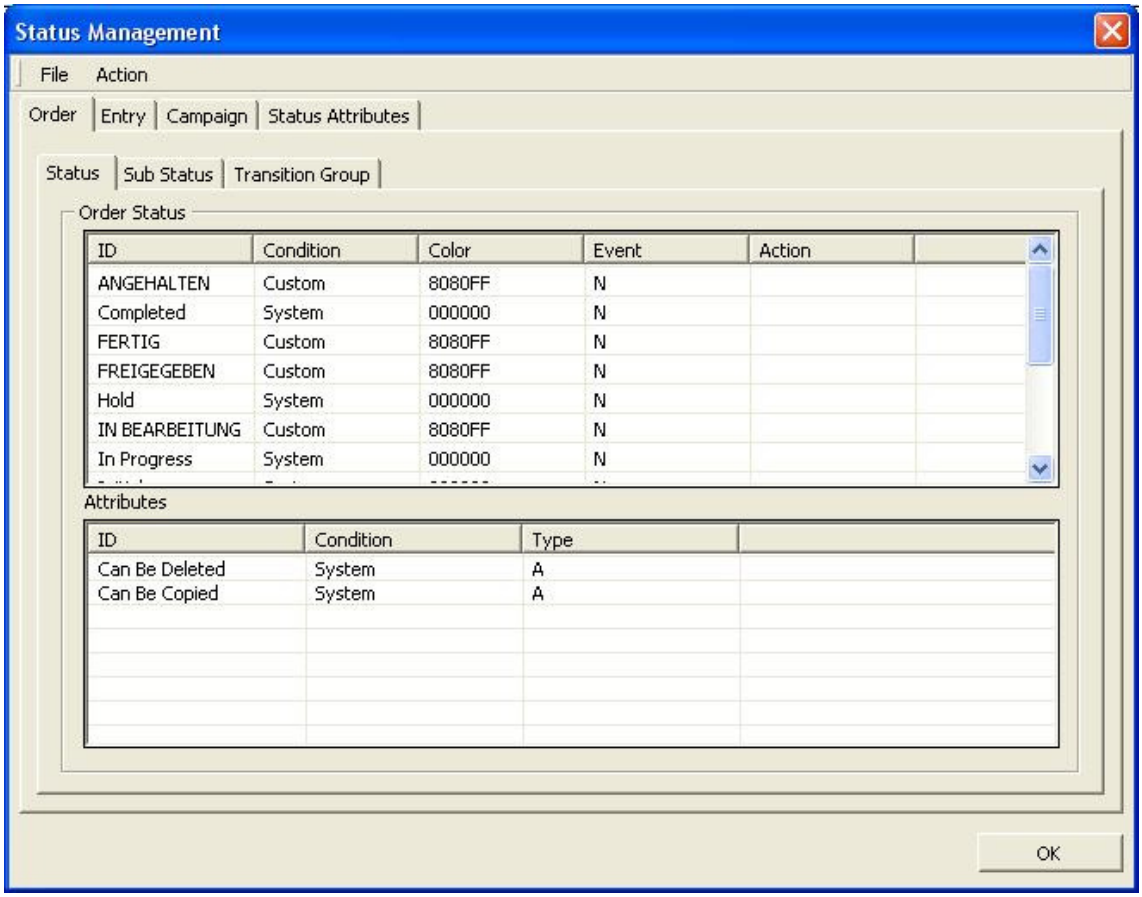

Abbildung 8.5: Statusmanagement

Um einen neuen Status zu erstellen, klickt man in der oberen Tabelle auf die rechte Maustaste und wählt dann „New…“ aus.

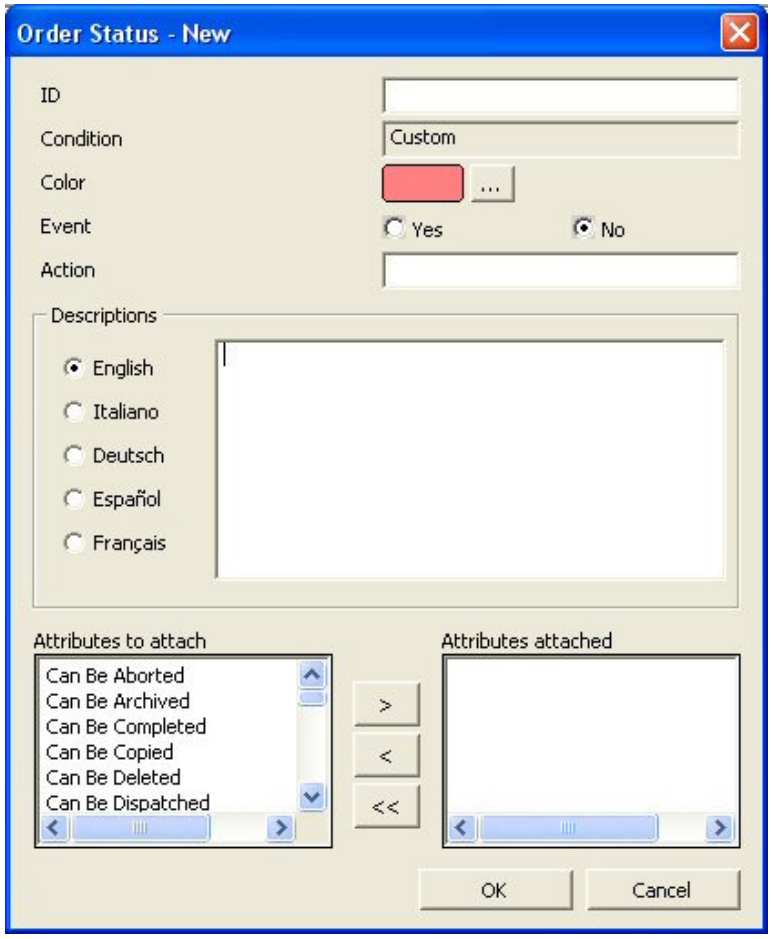

Abbildung 8.6: Eingabefenster zum Definieren eines neuen Status

Hier ist das wichtigste Eingabefeld „ID“. Bei „Description“ kann man noch Erklärungen in verschiedenen Sprachen hinterlegen, was aber optional ist. Im unteren Teil des Fensters kann man noch Attribute festlegen. Für den Simulationsbetrieb wurde hier immer „Can Be Copied“ und „Can Be Deleted“ eingegeben. Somit kann man einen Auftrag unabhängig vom Status löschen. Für einen realen Betrieb wäre dies nicht möglich, da sonst Aufträge und Auftragsdaten verloren gehen könnten.

Hat man nun alle Status definiert, muss man festlegen, von welchem Status man in einen anderen Status wechseln kann. Dies geschieht in so genannten Transition Groups (Übergangsgruppen). Es gibt voreingestellt die Transition Group „System“. Da hier aber mit selbst erstellten Zuständen gearbeitet wird, werden auch alle möglichen Zustandswechsel in einer eigenen Übergangsgruppe definiert. Dazu wählt man im Statusmanagementfenster die Registerkarte „Transition Group“ aus.

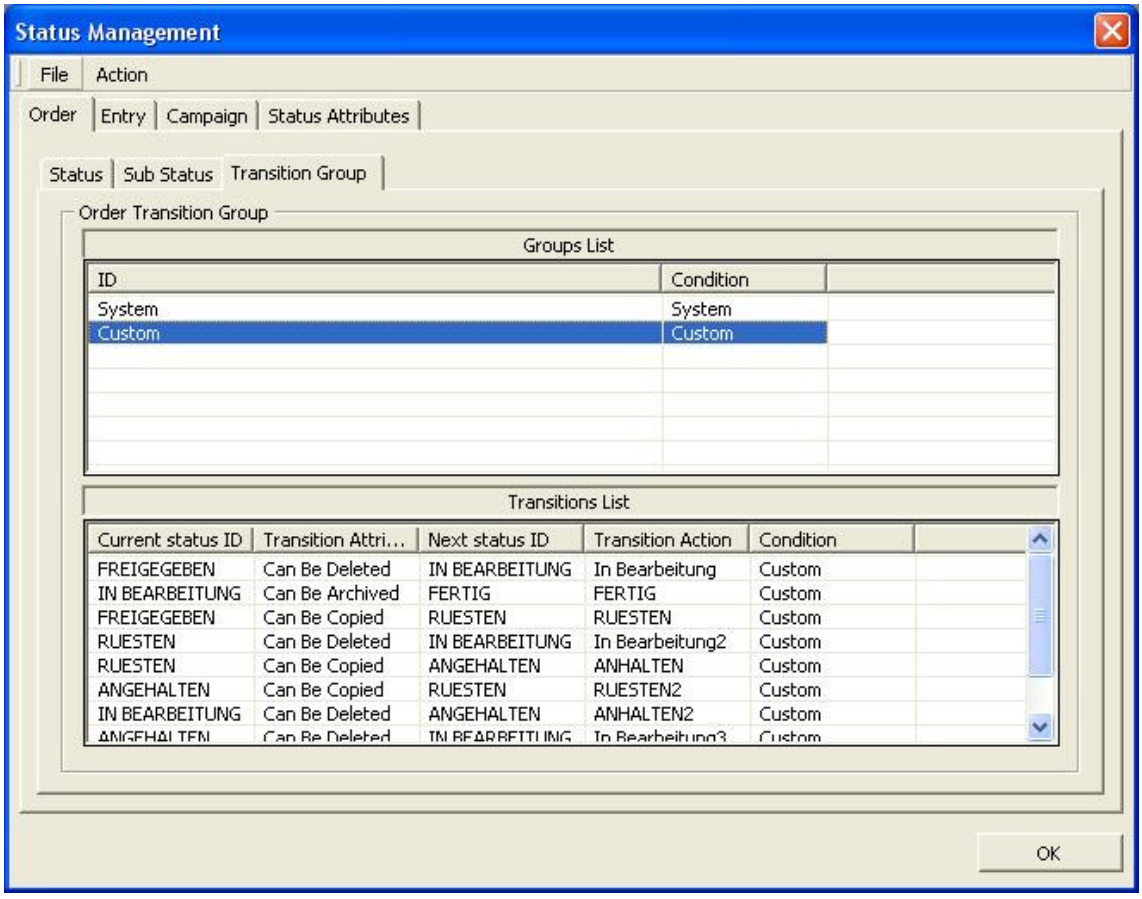

Abbildung 8.7: Übergangsgruppe

Zuerst erstellt man eine neue Übergangsgruppe, indem man in der oberen Tabelle mit der rechten Maustaste klickt und „New…“ auswählt. In dem sich nun öffnenden Fenster braucht man nur einen Namen für die neue Gruppe eingeben. Hier lautet die Bezeichnung der Gruppe „Custom“. Falls nötig, kann man noch eine Erklärung in verschiedenen Sprachen hinterlegen. Anschließend klickt man auf „OK“ und somit ist die neue Transition Group angelegt.

Man wählt diese aus und hat dann in der unteren Tabelle eine anfangs noch leere Liste. In dieser Liste befinden sich alle in der Übergangsgruppe definierten Zustandswechsel. Um einen neuen Wechsel hinzuzufügen, klickt man in der Tabelle wiederum rechts und wählt „New…“ aus und es öffnet sich folgendes Fenster:

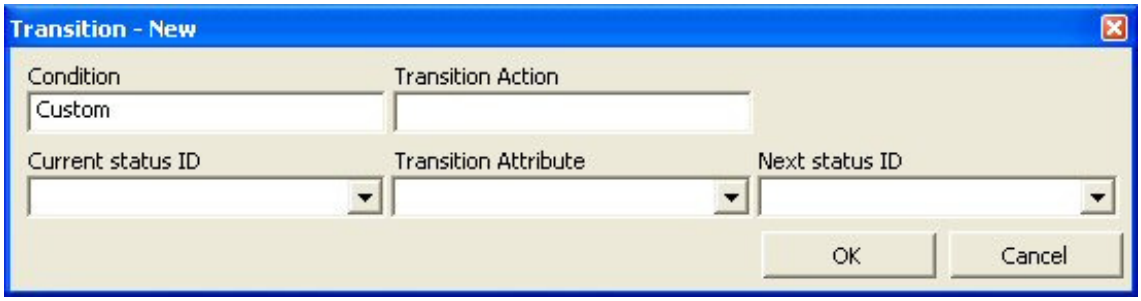

Abbildung 8.8: Statuswechsel definieren

Bei „Transition Action“ gibt man einen Namen für den Übergang ein. Aus der Drop-Down-Liste „Current status ID“ wählt man den Ausgangzustand und in der Drop-Down-Liste „Next status ID“ den Folgezustand aus. In der Drop-Down-Liste „Transition Attribute“ wählt man noch ein Attribut aus und klickt dann auf „OK“. Diesen Vorgang wiederholt man anschließend für alle möglichen Zustandswechsel.

8.1.3 XML-Struktur zum Importieren von Aufträgen

Damit Aufträge ins MES übernommen werden können, müssen diese in Form einer XML-Datei abgespeichert werden. XML heißt Extensible Markup Language. Diese Datei muss eine spezielle Struktur aufweisen, damit sie vom Production Modeler in den Production Order Manager übernommen werden kann. Ist diese Struktur fehlerhaft, so kommt die Fehlermeldung „XML-parsing failed“. Der folgende Text zeigt den Inhalt einer solchen XML-Datei. Normalerweise werden diese Dateien generiert, man kann sie aber auch selbst in einem Text-Editor oder einem speziellen XML-Editor erstellen.

```
<SCHEDPLAN XML_VERSION="4" DATETIME_FORMAT="yyyy-MM-dd
HH:mm:ss" DATE_SEP="-" TIME_SEP=":" TEMPLATE="NO">
  <ORDER>
    <ORDER_ID>ORDER_07</ORDER_ID>
    <PLANT_ID>DEMO_AREA-0</PLANT_ID>
    <STATUS_PK>1001</STATUS_PK>
    <TRANSITION_GROUP>1000</TRANSITION_GROUP>
    <TIMEFRAME_PK>2</TIMEFRAME_PK>
    <LAST_PROGRAM>POMD</LAST_PROGRAM>
    <ESTM_START_TIME>2007-08-22 03:00:00</ESTM_START_TIME>
    <ESTM_END_TIME></ESTM_END_TIME>
    <ORDER_DETAIL>
      <EQUIPMENT>DEMO_AREA-0</EQUIPMENT>
      <JOB>PROD_ACHSE</JOB>
      <OUTPUT_MATERIAL>ACHSE</OUTPUT_MATERIAL>
      <QUANTITY>5</QUANTITY>
        <BOM>
        <ID>BoM_Achse</ID>
        <VERSION>1</VERSION>
        <VERSION_NAME>1</VERSION_NAME>
        <CREATION_AUTHOR>1</CREATION_AUTHOR>
        <CREATION_TIMESTAMP>2007-08-10
03:00:00</CREATION_TIMESTAMP>
        <DEF_ID>BoM_Achse</DEF_ID>
        <QUANTITY>5</QUANTITY>
```

```
        <BOM_ITEM>
        <ID>Achse</ID>
        <DEF_ID>D12x50</DEF_ID>
        <SEQUENCE>1</SEQUENCE>
        <QUANTITY>1</QUANTITY>
        <BOM_UOM_ABBR>unit</BOM_UOM_ABBR>
        <CREATION_AUTHOR>1</CREATION_AUTHOR>
        <CREATION_TIMESTAMP>2007-08-10
03:00:00</CREATION_TIMESTAMP>
        </BOM_ITEM>
        </BOM>
        <CUSTOM_FIELD ID="NC-PROGRAMM">
          <VALUE></VALUE>
        </CUSTOM_FIELD>
      </ORDER_DETAIL>
      <ENTRY>
          <ENTRY_ID>DREHEN</ENTRY_ID>
          <EQUIPMENT>DEMO_AREA-0.DEMOANLAGE</EQUIPMENT>
          <JOB>DREHEN</JOB>
          <OUTPUT_MATERIAL>Achse</OUTPUT_MATERIAL>
          <QUANTITY>5</QUANTITY>
          <UOM>unit</UOM>
          <STATUS_PK>1000</STATUS_PK>
        <TRANSITION_GROUP>1000</TRANSITION_GROUP>
          <NOTE>NC-PROGRAMM</NOTE>
          <DISPATCH_MODE>Automatic</DISPATCH_MODE>
          <ESTM_START_TIME></ESTM_START_TIME>
          <ESTM_END_TIME></ESTM_END_TIME>
        <CUSTOM_FIELD ID="NC-PROGRAMM">
            <VALUE>test</VALUE>
          </CUSTOM_FIELD>
      </ENTRY>
  </ORDER>
</SCHEDPLAN>
```

Es ist wichtig, dass das XML-File folgende Beginnkennung besitzt:

```
<SCHEDPLAN XML_VERSION="4" DATETIME_FORMAT="yyyy-MM-dd
HH:mm:ss" DATE_SEP="-" TIME_SEP=":" TEMPLATE="NO">
```

Diese Beginnkennung legt eine „XML_VERSION“ fest, die bei uns immer „4“ ist. Danach wird noch die Schreibweise der Datums- und Zeitangaben in der Datei festgelegt. Jede Beginnkennung benötigt auch eine Endkennung. Diese sieht für die gesamte Datei wie folgt aus:

```
</SCHEDPLAN>
```

Zwischen diesen Kennungen können nun beliebig viele Aufträge nacheinander eingegeben werden. Ein Auftrag beginnt mit `<ORDER>` und endet mit `</ORDER>`. Dazwischen werden nun alle auftragsspezifischen Informationen eingegeben. Nach dem `</ORDER>` kann der nächste Auftrag begonnen werden.

Ähnlich wie bei `<SHEDPLAN>` und `<ORDER>` verhält es sich bei `<ORDER>` und <ENTRY>. Es kann also ein Auftrag mehrere Arbeitsgänge beinhalten.

Die meisten Begriffe im XML-File sind selbsterklärend. Für die anderen ist hier eine kurze Erläuterung angeführt:

- STATUS_PK: Diese Zahl (Primary Key) legt fest, welchen Status ein Auftrag haben soll. Jeder in POM erstellte Status hat eine STATUS_PK-Nummer. Diese findet man in der „SITMesDb“-Datenbank des SQL-Servers. Für den Auftrag findet man die STATUS_PK in der Tabelle POM_ORDER_STATUS und für den Arbeitsgang in der Tabelle POM_ENTRY_STATUS.
- TRANSITION_GROUP: Damit Aufträge und Arbeitsgänge den Status wechseln können, müssen die möglichen Zustandswechsel definiert werden. Diese Definitionen befinden sich in einer TRANSITION_GROUP (Übergangsgruppe). Diese Gruppen haben eine Identität wie hier zum Beispiel „Custom“. Die XML-Datei benötigt aber nicht die Identität, sondern den zugehörigen Primary Key. Dieser befindet sich in der „SITMesDb“ in der Tabelle POM_ORDER_TRANSITION_GROUP für Aufträge bzw. POM_ENTRY_TRANSITION_GROUP für Arbeitsgänge.
- BOM: Für jeden Order und Entry kann hier eine auftragsbezogene Stückliste eingefügt werden. Im obigen Beispiel gibt es nur eine Auftragsstückliste. Die Stückliste für einen Arbeitsgang sieht aber ident aus und wird ebenfalls vor einem eventuellen Custom Field eingefügt.
- CUSTOM_FIELDS: Ein Auftrag hat viele vordefinierte Eingabemöglichkeiten. Manchmal ist es aber notwendig, mehr Information mit dem Auftrag zu übergeben. Ein Beispiel dazu ist ein NC-Programm, welches die Maschine für den Arbeitsgang benötigt. Diese Benutzerfelder (Custom Fields) werden im Auftrag vor dem ersten Arbeitsgang und beim Arbeitsgang ganz zum Schluss eingefügt. Ein Benutzerfeld sieht wie folgt aus:

```
<CUSTOM_FIELD ID="NC-PROGRAMM">
  <VALUE>test</VALUE>
</CUSTOM_FIELD>
```

In der Beginnkennung steht die Identität und im Tag „Value“ befindet sich der eigentliche Inhalt. Wie man solche Custom Fields definiert, wird in 8.1.3.1 beschrieben.
Soll ein Auftrag oder Arbeitsgang ein Custom Field enthalten, dessen Inhalt erst später eingefügt wird, lässt man den Tag „Value“ einfach leer. Würde man beim Erstellen des Auftrages das Benutzerfeld ganz weglassen, wäre es schwieriger, es nachträglich einzufügen.

8.1.3.1 Erstellen eines Benutzerfeldes

Benutzerfelder oder Custom Fields sind, wie bereits oben erklärt, benutzerdefinierte Informationen, die ein Auftrag oder Arbeitsgang enthalten kann. Bevor ein solches Custom Field einem Auftrag zugewiesen werden kann, muss es erst im Production Order Manager definiert werden. Man geht also in der Menüleiste des POM auf Tools – Custom Fields – Management und es öffnet sich folgendes Fenster.

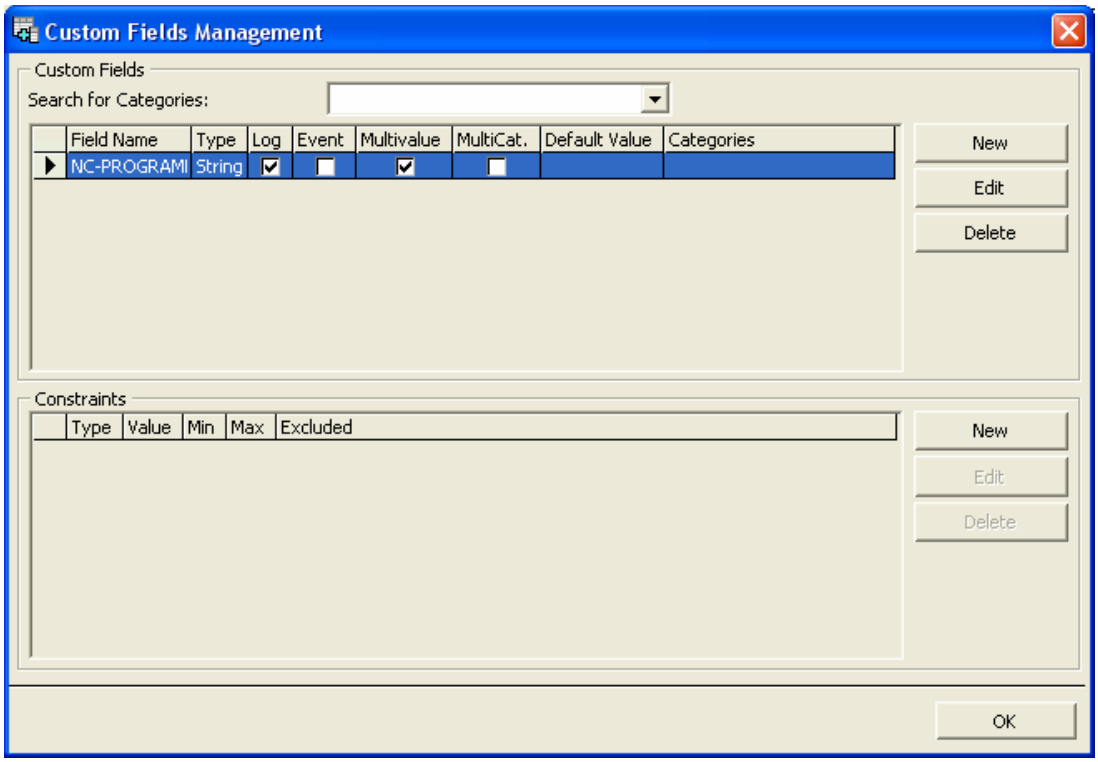

Abbildung 8.9: Erstellen eines Benutzerfeldes

In diesem Feld klickt man auf den Button „New“ und gelangt somit zur Eingabemaske für ein neues Custom Field. Dort sind die wichtigsten Felder „Field Name“ und „Type“. Die anderen Felder können leer gelassen werden. Manchmal ist es jedoch sinnvoll, noch die Kontrollkästchen „Log“ und „Multivalue“ auszuwählen. Wenn „Log“ aktiviert ist, werden alle getätigten Eingaben im Hintergrund mitprotokolliert und können bei einer späteren Auswertung hilfreich sein. „Multivalue“ heißt, dass im Benutzerfeld mehrere Werte abgespeichert werden können.

8.2 Bedienung des Material Managers (MM)

Allgemeines zum Material Manager wurde bereits in Kapitel 5.5 erklärt. In diesem Teil wird deshalb nur mehr die Bedienung und Handhabung der Material Manager Benutzeroberfläche beschrieben. Im Material Manager sind alle Materialien, deren Menge und deren Lager- bzw. Aufenthaltsort hinterlegt. Man kann die Materialien hier also verwalten. In diesem Projekt wird die Material Manager Benutzeroberfläche nur benutzt, um Roh- bzw. Fertigmaterialien zu erstellen. Darum sind auch nur diese Anwendungsschritte näher erklärt.

Die Disposition und Verwaltung geschieht über den Production Modeler, der über das Global-Settings-Interface mit dem MM kommuniziert und dort Methoden aufruft. Falls im PM Fehler auftreten, können diese hier leicht ausgebessert werden. Dies ist jedoch nicht näher beschrieben. Die Hilfe das Material Managers ist hier ausreichend.

Öffnet man den Material Manager Presentation Client wird folgendes Ausgangsfenster sichtbar.

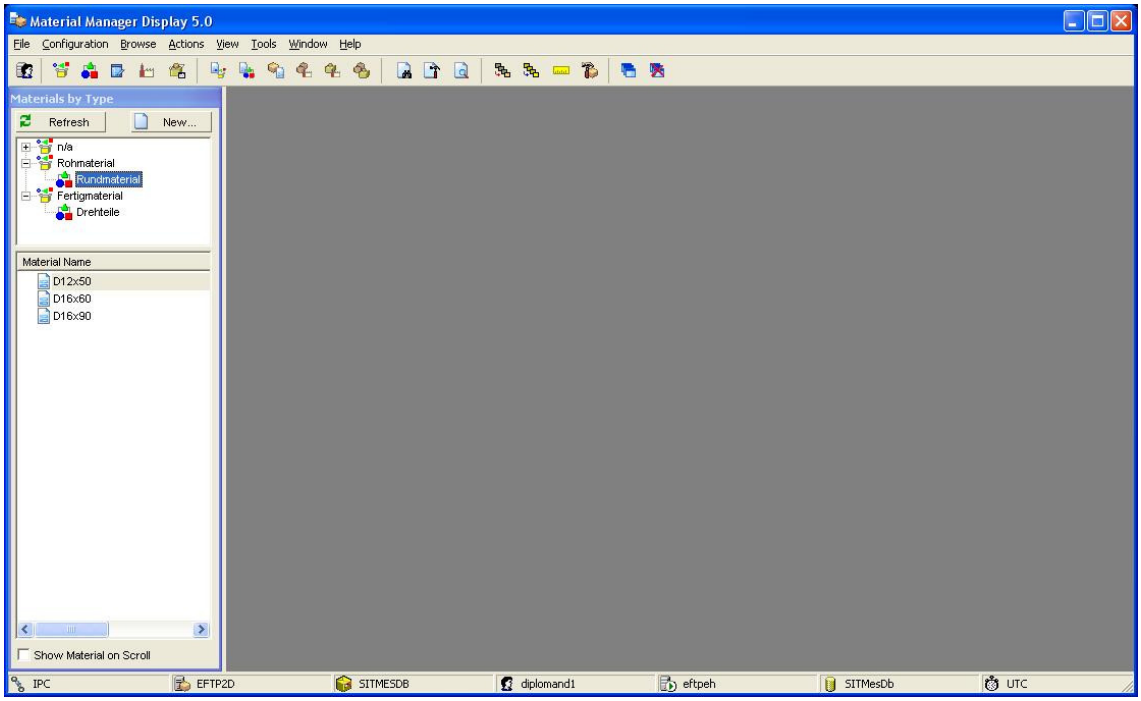

Abbildung 8.10: Material Manager Display

Im linken Teil von Abbildung 8.10 ist in einer Baumstruktur das Typenmanagement ersichtlich. Es sind die Materialtypen „Rohmaterial“ und „Fertigmaterial“ zu sehen. Diese Typen erstellt man, indem man in der Menüleiste Configuration – Type Management auswählt. Es wird nun folgendes Fenster angezeigt.

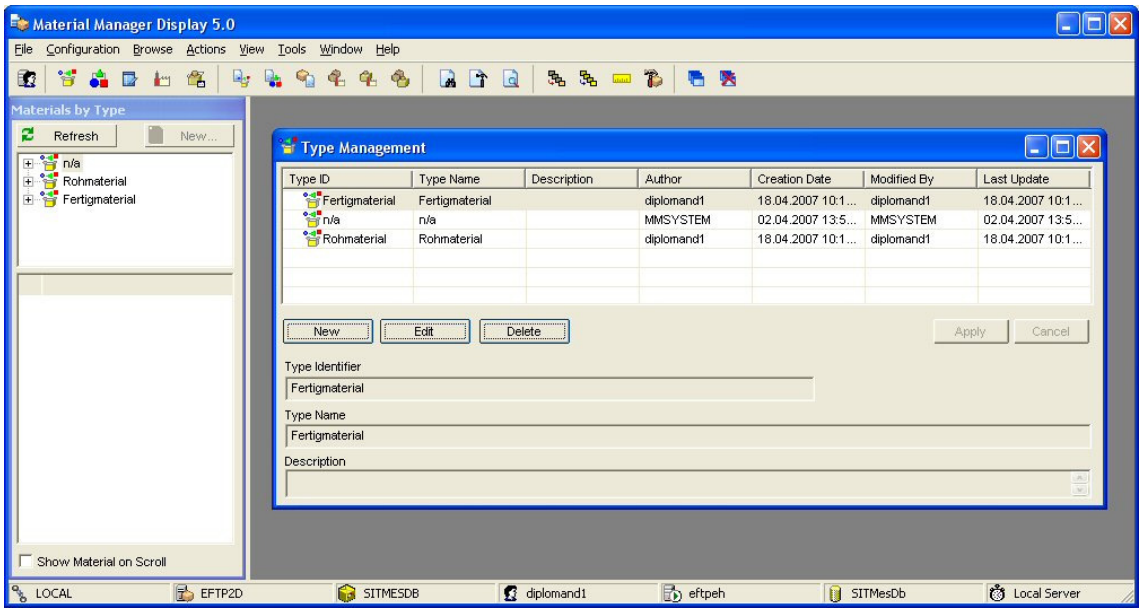

Abbildung 8.11: Type Management

Um einen neuen Materialtyp zu erstellen, klickt man auf „New“ und füllt dann die Felder

- Type Identifier,
- Type Management und
- Description aus.

Das Feld „Description“ ist nicht obligatorisch. Sind die Eingaben getätigt, braucht man nur mehr auf „Apply“ zu klicken. Für eventuelle Änderungen braucht man nur den „Edit“-Button drücken und kann die Eingabefelder bearbeiten.

Wenn die Typen erstellt sind, ist es notwendig Materialklassen zu erstellen. Das Bearbeitungszentrum in diesem Projekt ist ein Drehautomat und deshalb wird bei dem Typ Rohmaterial die Klasse „Rundmaterial“ und beim Typ „Fertigmaterial“ die Klasse Drehteile hinterlegt. Man geht hier ähnlich vor wie bei der Typenerstellung. Man wählt in der Menüleiste Configuration – Class Management aus. Das Fenster in Abbildung 8.12 wird angezeigt. Um eine neue Klasse zu erstellen, klickt man wiederum auf „New“ und füllt die Eingabefelder aus. Um den „Material Type“ einzugeben, muss man auf die Schaltfläche neben dem Eingabefeld klicken und eine zuvor erstellte Type auswählen.

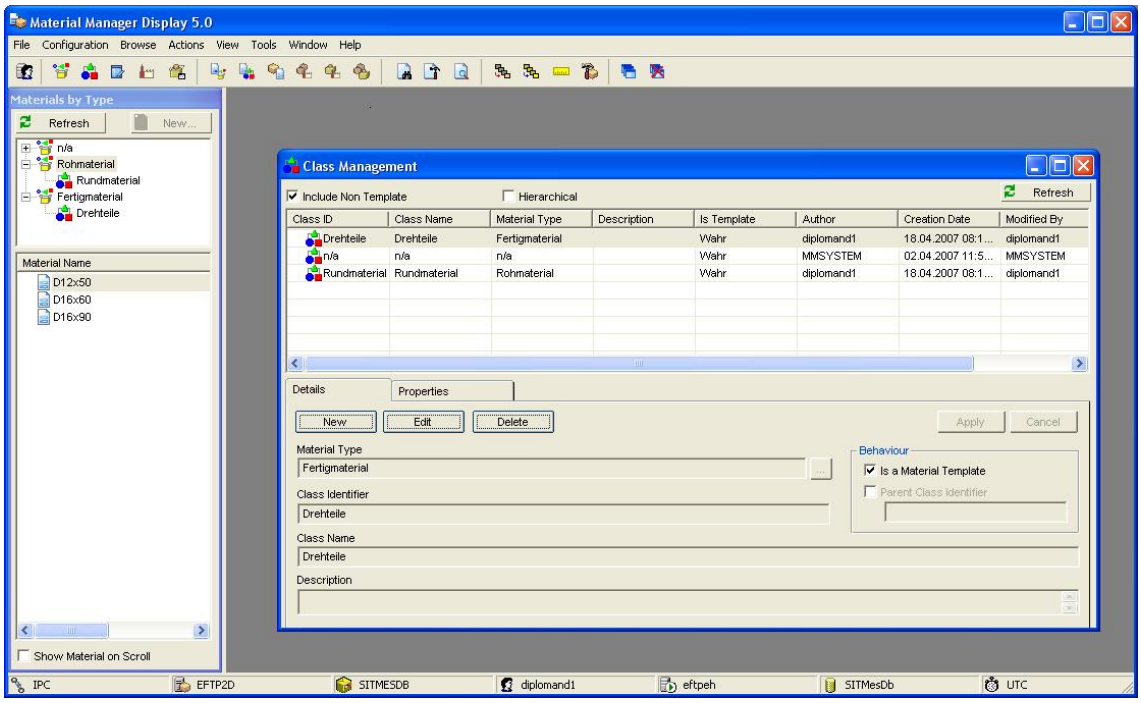

Abbildung 8.12: Class Management

Sind diese Tätigkeiten abgeschlossen, kann man nun die eigentlichen Materialien erstellen. Dazu wählt man im Material Manager Display in der Baumstruktur den gewünschten Typ und eine Materialklasse aus. Anschließend klickt man auf den Button „New“ über der Baumstruktur und folgendes Fenster wird angezeigt.

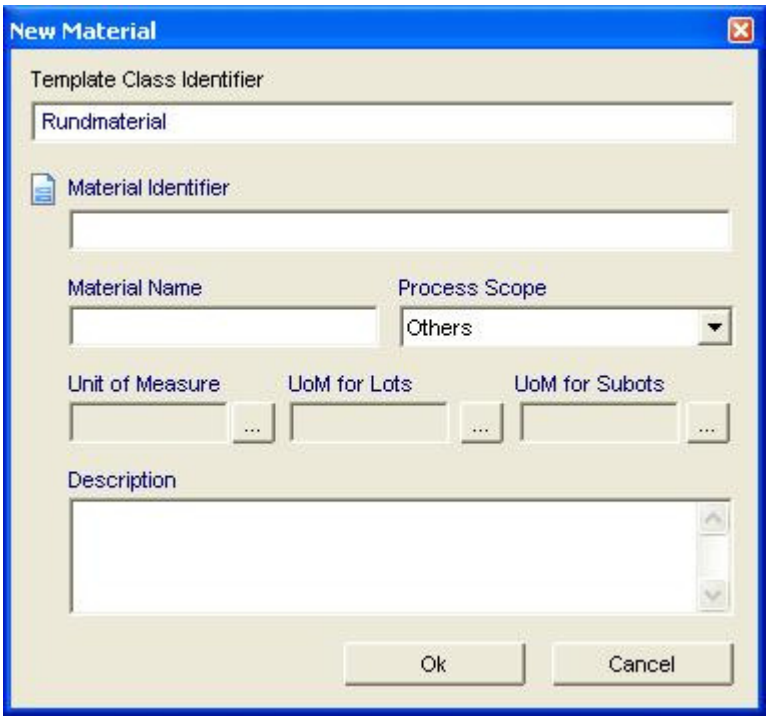

Abbildung 8.13: New Material

Im obigen Fenster kann man alle Eingaben für ein neues Material tätigen.

- Template Class Identifier: Hier wird die zuvor gewählte Materialklasse angezeigt.
- Material Identifier: Hier gibt man eine genaue Bezeichnung für das Material ein (z.B. eine Materialnummer). In diesem Projekt wird aber hier die gleiche Bezeichnung wie beim Materialnamen eingegeben.
- Material Name: Der Materialname muss in diesem Feld eingegeben werden (z.B. D12x60).
- Process Scope: Hier wird ein Prozessbereich eingegeben. In diesem Fall wählt man in der Drop-Down-Liste „Discrete Manufacturing“ aus.
- Unit of Measure: Bei den einzelnen UoM-Feldern wird in diesem Projekt immer „Unit“, also „Stück“ ausgewählt. Man kann hier aber auch Menge oder Gewicht auswählen.
- Description: Eine Beschreibung muss nicht unbedingt eingegeben werden.

Es werden nun alle erforderlichen Materialien erstellt. Es sind aber noch keine Lose oder Ähnliches vorhanden. In diesem Projekt werden MM-Methoden zum Erstellen von Handling-Units, Losen und Sublosen über den PM aufgerufen. Es sind also keine weiteren Schritte in der MM-Benutzeroberfläche durchzuführen.

8.3 Bedienung des Data Integration Service

8.3.1 Aufgabenstellung und Allgemeines

Das Data Integration Service übernimmt in dieser Diplomarbeit die Schnittstelle zwischen dem Scheduler und dem MES. Der Scheduler kopiert freigegebene Aufträge im XML-Format in einen vordefinierten Ordner. Das Data Integration Service muss dieses File sofort nach dessen Erstellung abrufen und den Auftrag in den Server speichern. Danach sendet der Server eine Nachricht über den Erhalt eines oder mehrerer neuer Fertigungsaufträge an den Production Modeler. Der PM ruft die Aufträge dann vom DIS-Server ab und schreibt die enthaltenen Aufträge in den POM. Damit diese Aufgaben durchgeführt werden können, stellt die DIS Management Console Connectoren zur Verfügung. Das Bedienen der Management Console und das Erstellen der Connectoren sind im folgenden Teil beschrieben.

8.3.2 Bedienung der DIS Management Console

Nach dem öffnen der DIS Management Console erscheint das Fenster in Abbildung 8.14. Im linken Teil des Fensters sind alle erstellten Projekte zu sehen. Es kann immer nur ein Projekt aktiv sein. Alle benötigten Connectoren müssen in diesem Projekt gespeichert sein. Das aktive Projekt ist in der Management Console fett gedruckt.

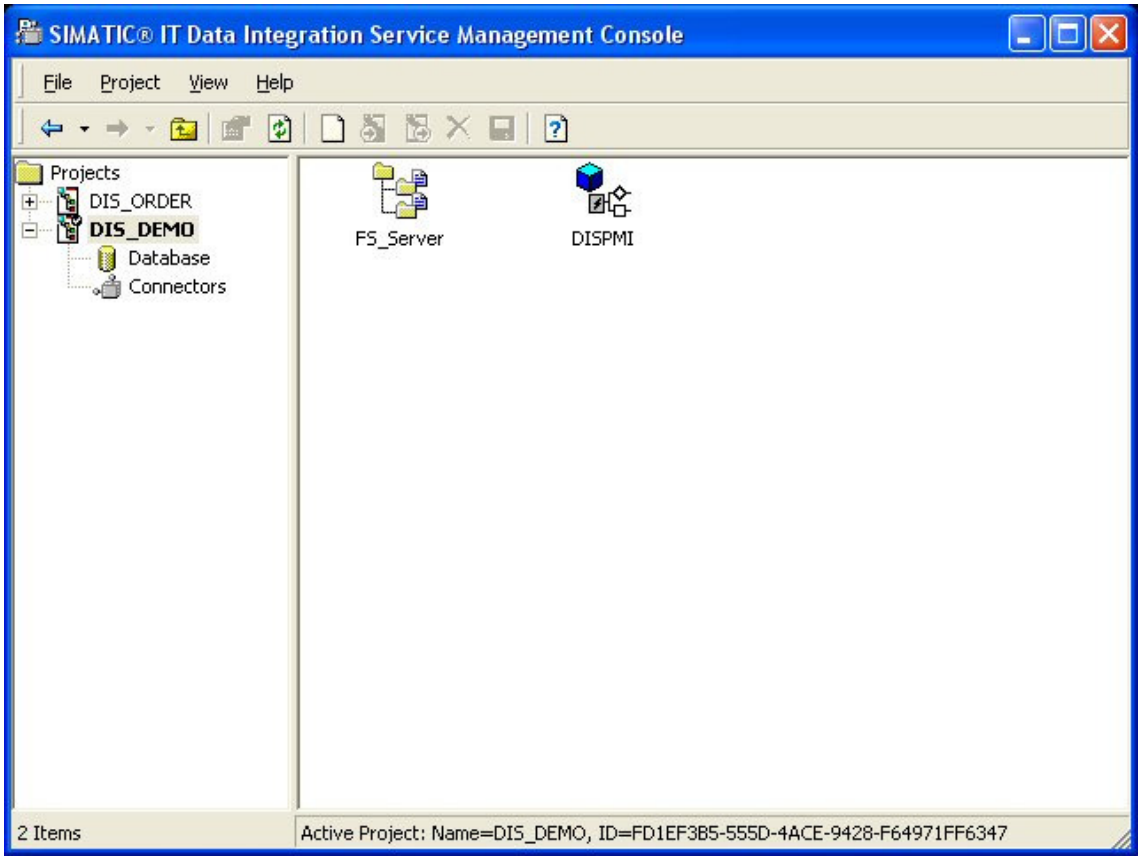

Abbildung 8.14: DIS Management Console

8.3.2.1 Erstellen eines neuen Projekts

Um ein neues Projekt zu erstellen, geht man in der Menüleiste auf File und wählt „New Project“ aus. Es erscheint dann folgendes Fenster.

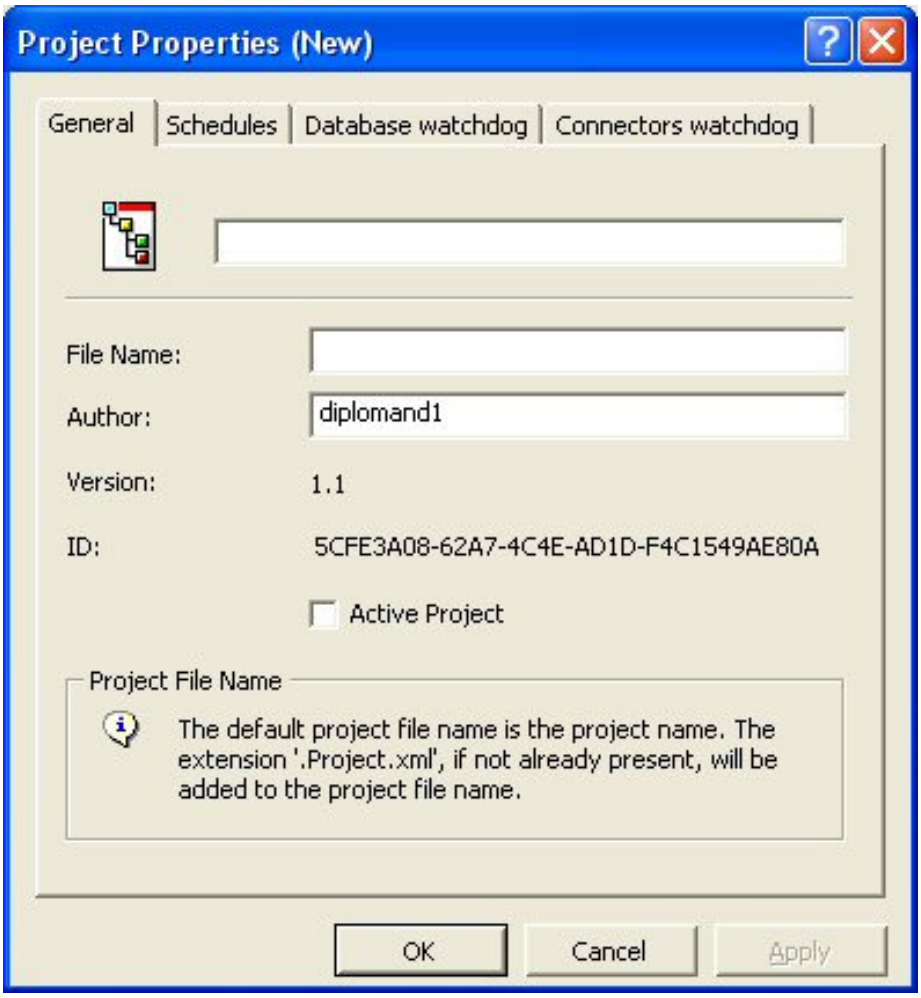

Abbildung 8.15: Projekteigenschaften

Hier benötigt man nur eine Bezeichnung und einen „File-Name“.

- Project Name: `DIS_DEMO`
- File Name: `DIS_DEMO.Project.xml`

Sind diese Eingaben getätigt, muss man das Feld „Active Projekt“ setzen und mit „OK“ bestätigen. Das Projekt ist nun erstellt.

8.3.2.2 Erstellen der Connectoren

Als nächster Schritt müssen dann die Connectoren erstellt werden. Es werden zwei Connectoren benötigt. Diese sind ein File-System-Server und ein PM-Connector.

- Der File-System-Server ist dafür zuständig, die vom ERP abgespeicherten Fertigungsaufträge in den DIS-Server zu übernehmen. Um einen neuen Connector

zu erstellen, wählt man in der Menüleiste Project – Add Connector – File-System-Server aus. Es erscheint anschließend eine Eingabemaske (Abbildung 8.16).

Abbildung 8.16: File-System-Server Registerkarte „General“

In der Registerkarte „General“ braucht man nur einen Namen einzugeben. In diesem Fall ist das FS_Server. Danach wählt man die Registerkarte „Incoming Messages“ (Abbildung 8.17).

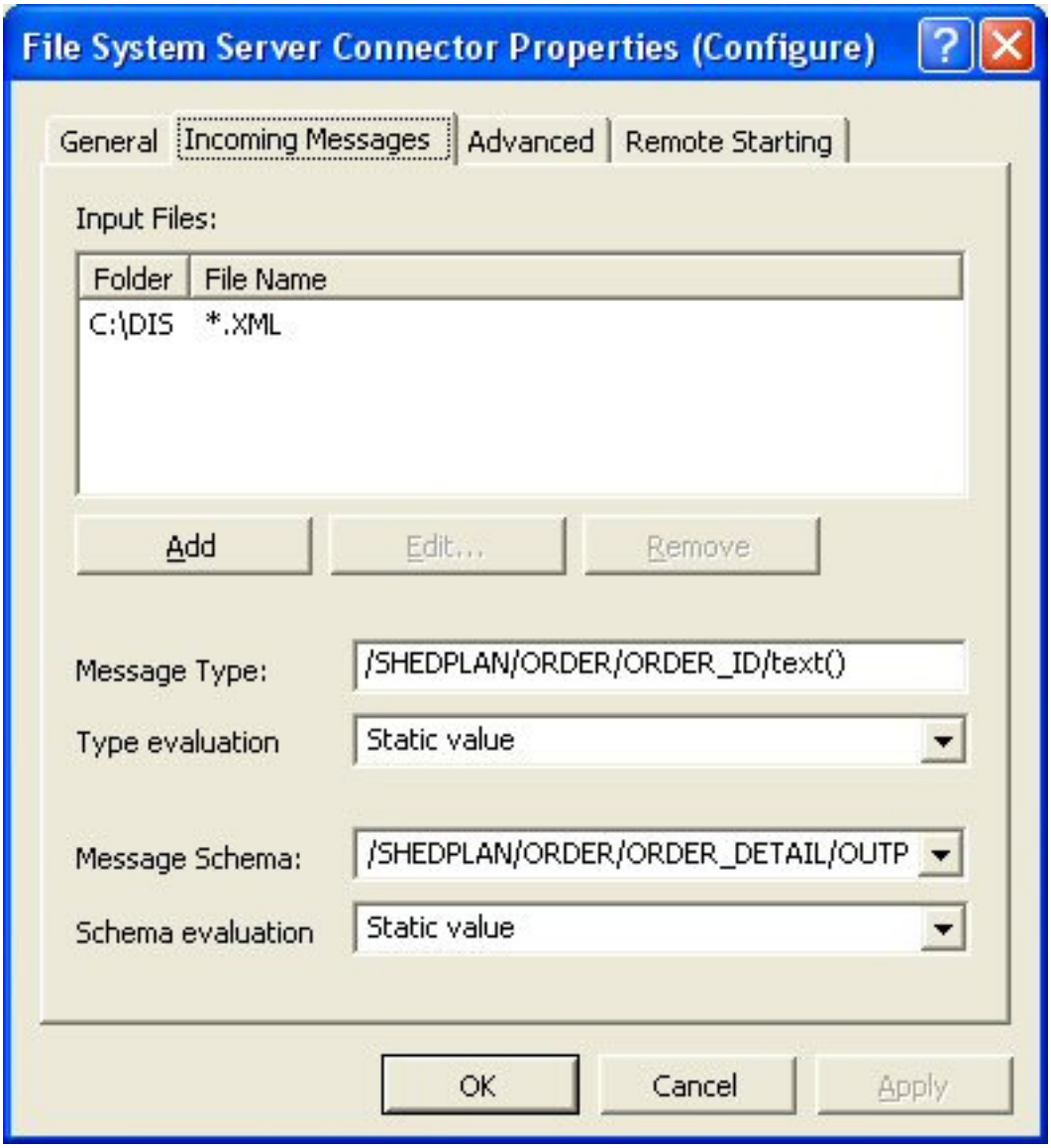

Abbildung 8.17: File-System-Server Registerkarte „Incoming Messages“

In dieser Registerkarte legt man den Ordner fest, der vom Server wegen neuer Aufträge überwacht wird. Man klickt dazu auf „Add“ und gibt dann den Ordnerpfad an. Danach muss man noch angeben, dass XML-Dateien eingelesen werden sollen. Dies geschieht mit „`*.xml`“.

Es sind dann noch ein „Message Type“ und ein „Message Schema“ festzulegen. Typ und Schema sind Filter, die überprüfen sollen, ob das File ein gültiges XML-Übergabefile ist.

„Message Type“ überprüft, ob eine „ORDER_ID“ vorhanden ist und sieht wie folgt aus:
`/SHEDPLAN/ORDER/ORDER_ID/text()`

„Message Schema“ überprüft noch ob ein „OUTPUT_MATERIAL“ vorhanden ist und sieht wie folgt aus:
`/SHEDPLAN/ORDER/ORDER_DETAIL/OUTPUT_MATERIAL/text()`.

In der Registerkarte Advanced kann man anschließend noch einen Ordner festlegen, in den ungültige Dateien kopiert werden.

- Der PM-Connector ist für die Verbindung zum Production Modeler zuständig. Er benachrichtigt den PM, wenn ein neuer Auftrag in den Server eingegangen ist und übermittelt diesen, wenn er vom Production Modeler angefordert wird. Beim Erstellen ist die Vorgangsweise ähnlich wie zuvor. Man wählt nur statt einem File-System-Server einen PM-Connector aus. Es erscheint dann folgendes Fenster.

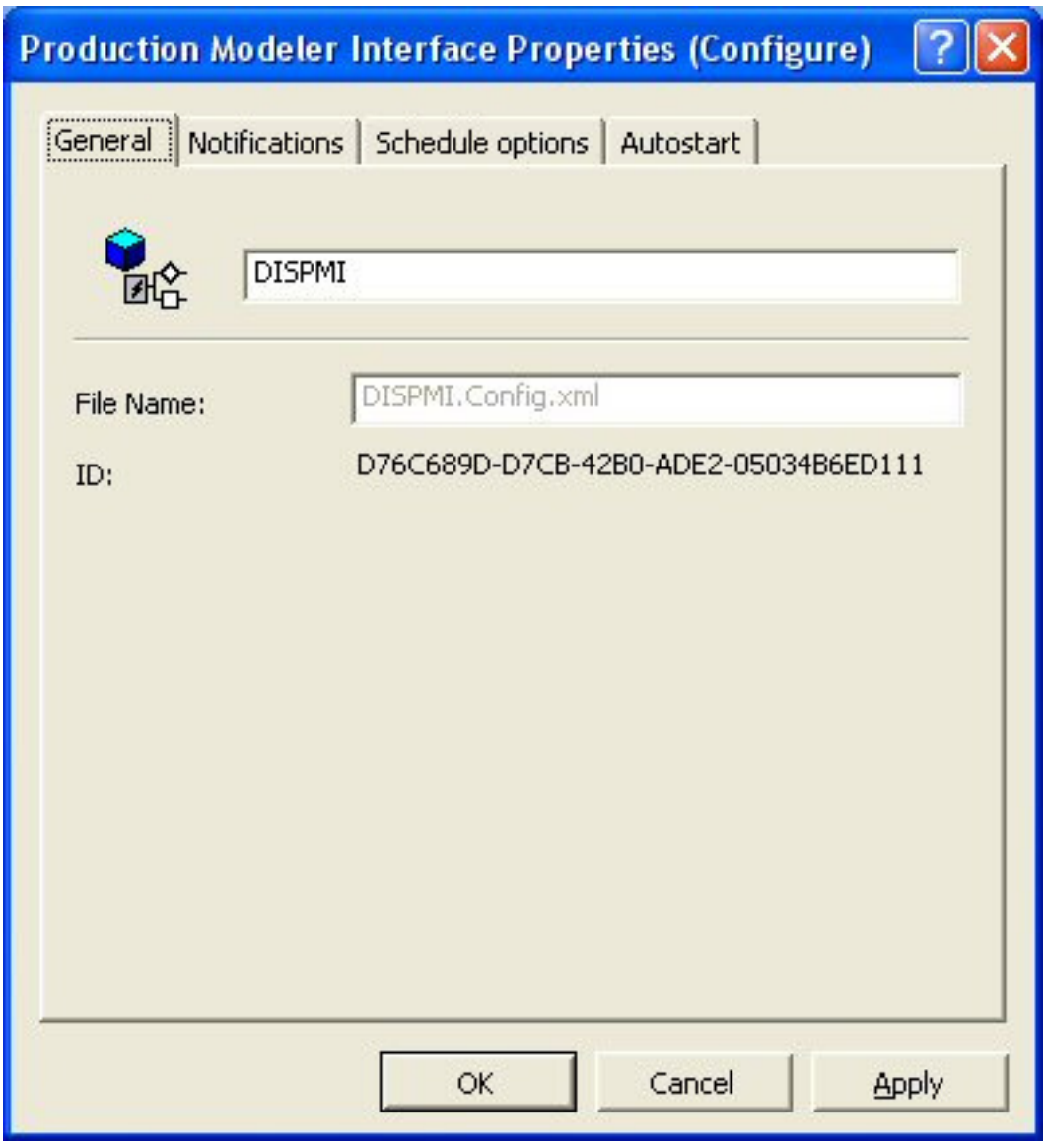

Abbildung 8.18: PM-Connector

In der Registerkarte „General“ ist nur ein Name und Dateiname einzugeben. Die Eingaben können aus Abbildung 8.18 entnommen werden.

In der Registerkarte „Notifications“ wird festgelegt, welche Nachrichten an den PM gesendet werden. Man definiert Typ und Schema und es werden dann nur Nachrichten mit dem definierten Typ und Schema an den PM gemeldet.

Um Typ und Schema zu definieren, klickt man in der Registerkarte „Notifications“ auf „Add“. In der sich öffnenden Eingabemaske kann man Typ und Schema eingeben. Bei diesem PM-Connector sollen aber alle Nachrichten, die einen „Message Type“ und ein „Message Schema“ besitzen, an den PM übermittelt werden. Aus diesem Grund gibt man in beide Felder nur „*“ ein.

Somit sind alle Arbeiten in der SIMATIC IT DIS Management Console abgeschlossen und es kann aus dem Production Modeler damit gearbeitet werden.

9 Programmierung der Bedieneroberfläche für die Fertigungszelle

9.1 Aufgabenstellung

Der Bediener der Anlage in der Fertigungsebene soll eine geeignete Benutzeroberfläche erhalten, um die Arbeitsgänge der einzelnen Aufträge zu verwalten, sowie mit Hilfe eines Handbetriebes bei Störungen in die Anlage einzugreifen. Der Bediener soll Arbeitsgänge an der Anlage anmelden, Leerpaletten in die Anlage einlegen und Paletten, die nicht mehr benötigt werden, aus der Anlage entnehmen können. Weiters sollen Zeitauswertungen möglich gemacht werden sowie eine Übersicht, wo sich verschiedene Lose und Sublose befinden. Die Oberfläche soll möglichst einfach gestaltet und über ein Menü zu handhaben sein. Die Struktur der Bedieneroberfläche sehen sie in der folgenden Abbildung.

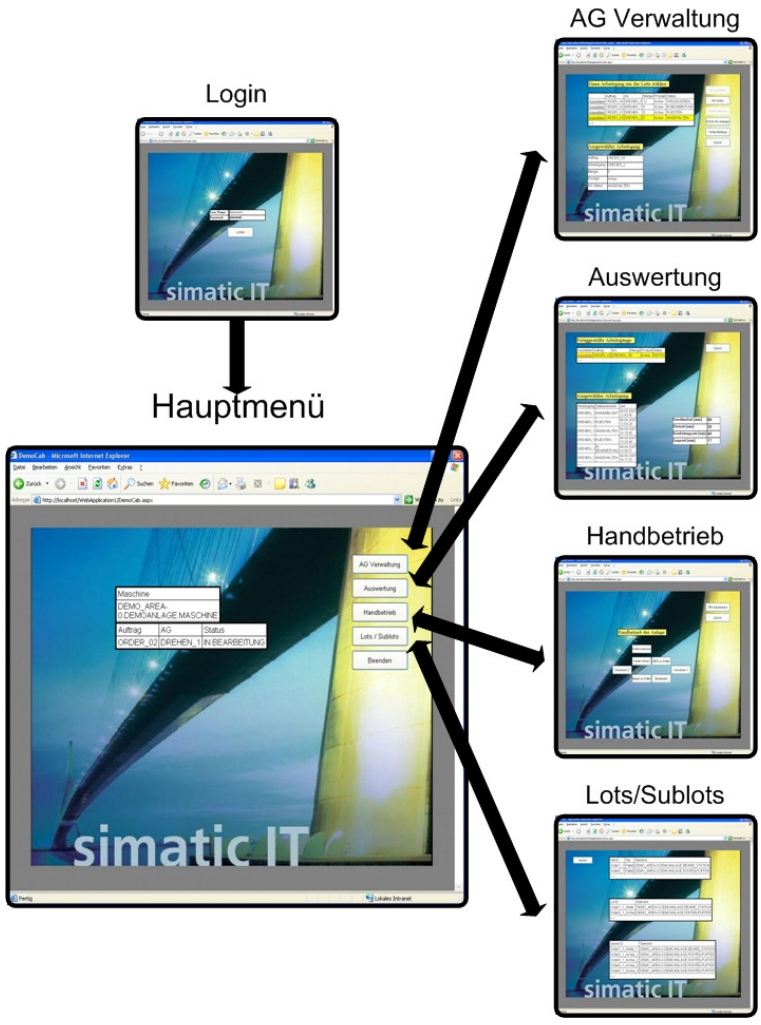

Abbildung 9.1: Struktur Bedieneroberfläche

9.2 Allgemeines

Um eine geeignete Oberfläche für den Bediener der Fertigungszelle zu programmieren stellt SIMATIC IT Production Suite die CAB Funktionen zur Verfügung. CAB ist die Abkürzung für Client Application Builder und ist eine Technologie, welche in die Programmiersoftware Microsoft Visual Studios.NET 2003 eingebunden wird. Die Verwendung von ASP (Active Server Pages) ist unumgänglich, da die CAB Anwendungen nur für Web-Formulare funktionsfähig sind.

> *„Microsoft ASP.NET besteht aus einer Reihe von Technologien zur Entwicklung von Webanwendungen, die es Entwicklern ermöglichen, dynamische Webseiten, Webanwendungen und XML-Webdienste zu erstellen.“*[8].

Web-Anwendungen sind Anwendungen, die von einem Webbrowser (z.B.: MS Internet Explorer) gestartet beziehungsweise ausgeführt werden können. Die Seiten einer Web-Anwendung heißen Web-Formulare (.ASPX) und können mit der Software MS Visual Studios.NET modelliert werden. Nähere Informationen zum Thema MS Visual Basic.NET und ASP.NET ist in einschlägiger Fachliteratur nachzulesen, siehe Literaturverzeichnis.

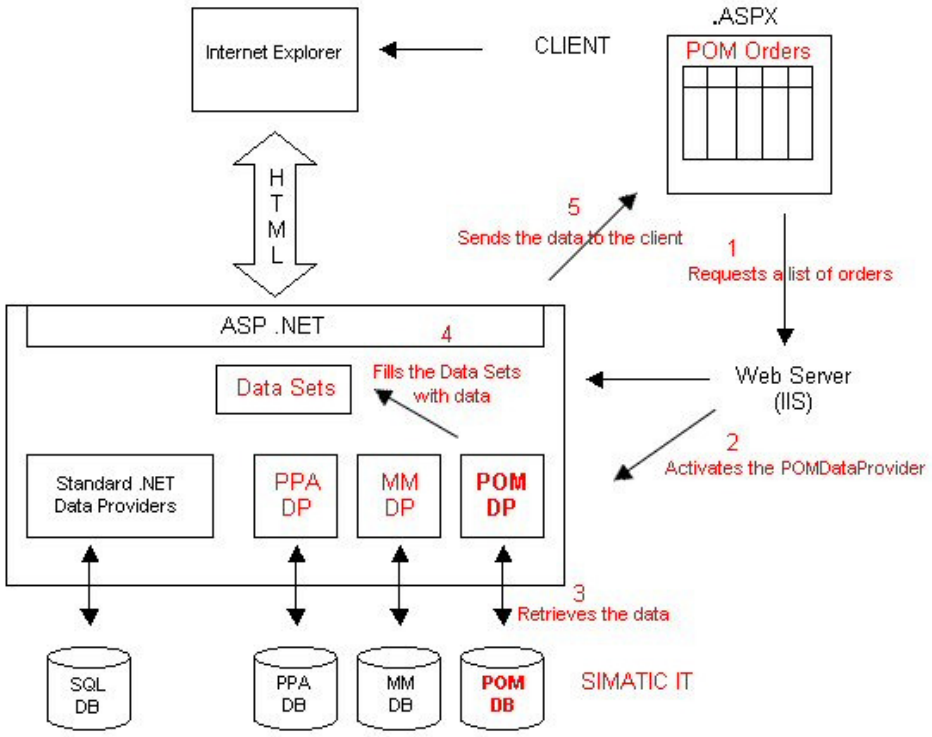

Abbildung 9.2: Aufbau eines Web-Formulars mit ASP[9]

[8] URL: http://msdn2.microsoft.com/de-de/asp.net/default.aspx [05.11.2007]

[9] Quelle: Siemens AG (2005).

9.3 Erstellen einer Web-Anwendung mit MS Visual Studio

Nach dem Start von MS Visual Studio muss eine neue Web-Anwendung geöffnet werden. Zu diesem Zweck drückt man in der Menüleiste auf Datei – Neu – Projekt und ein Fenster mit dem Namen „Neues Projekt“ wird geöffnet , siehe Abbildung 9.3.

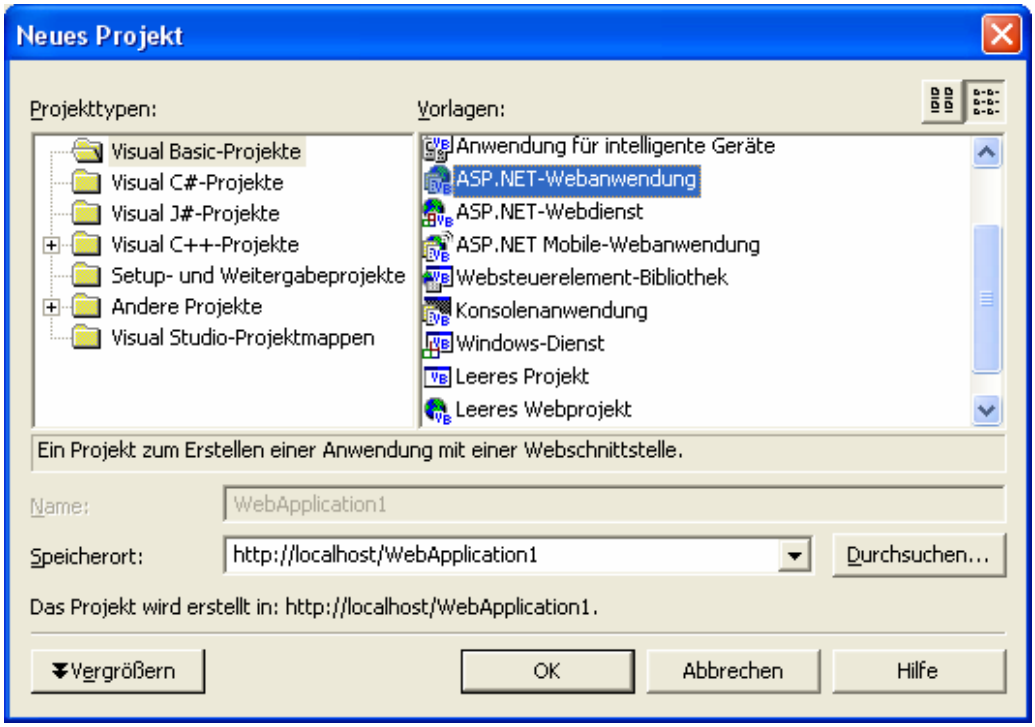

Abbildung 9.3: Neues Projekt in MS Visual Studios.NET erstellen

Unter Projekttypen (linker Bereich des Fensters) wählt man Visual Basic-Projekte aus und findet dann im rechten Bereich des Fensters unter Vorlagen eine Reihe von möglichen Anwendungen. Um die CAB Funktionen nutzen zu können muss eine „ASP.NET-Web-Anwendung“ ausgewählt werden und durch bestätigen mit dem „OK“-Button wird eine solche geöffnet. Im Eingabefeld „Speicherort“ kann der Name festgelegt werden, unter dem das Projekt gespeichert werden soll, damit spart man sich eine spätere Umbenennung.

9.3.1 Laden der CAB Toolbox

SIMATIC IT Client Application Builder ist eine Entwicklungsplattform, welche dem Anwender erlaubt graphische Oberflächen zu erstellen, auf denen spezifische Daten des MES angezeigt und Funktionen gestartet werden können.

Zu diesem Zweck muss man in Microsoft Visual Basic.NET eine zusätzliche Bibliothek laden, siehe Abbildung 9.4. In der Menüleiste befindet sich ein zusätzliches Menü mit dem Namen CAB, in dem diese Einstellungen vorgenommen werden können. CAB ist also ein Plug-In für Microsoft Visual Basic.NET.

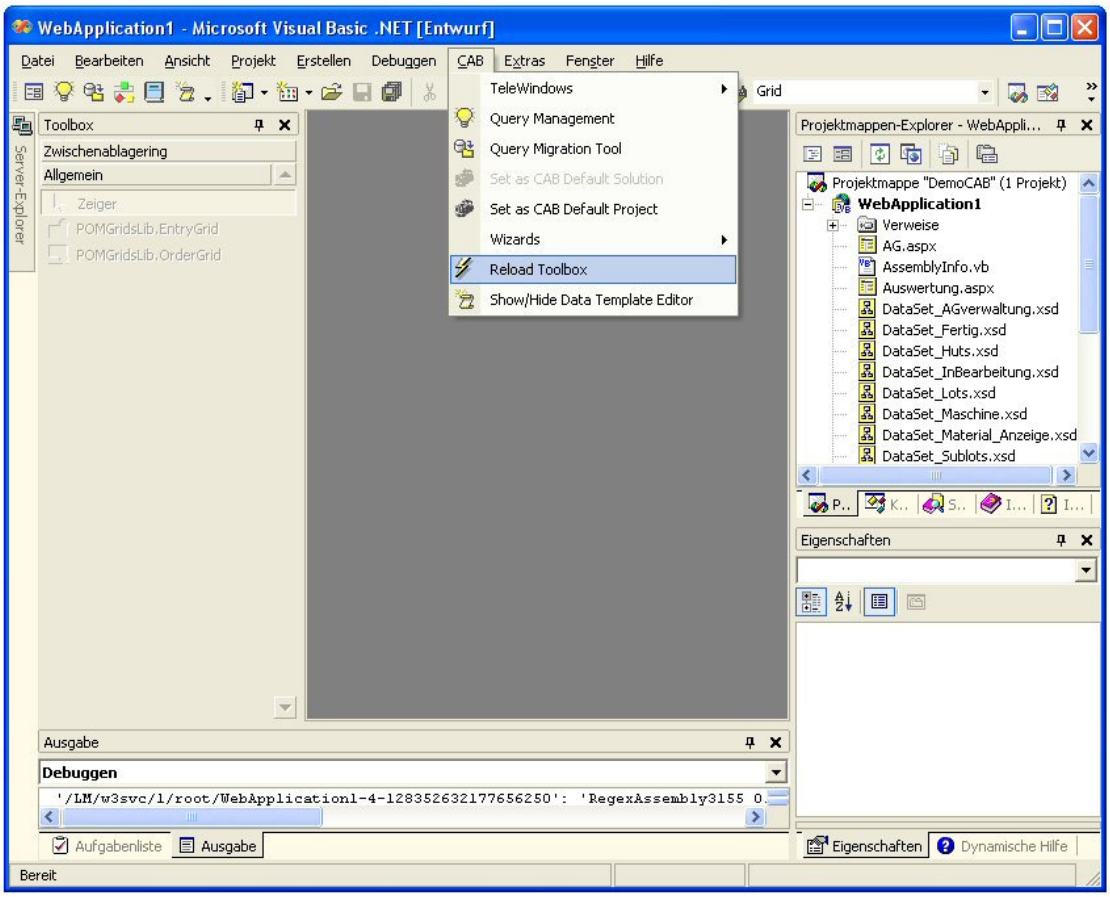

Abbildung 9.4: Laden der CAB Bibliotheken in MS Visual Basic.NET

Wenn das Toolbox-Fenster nicht angezeigt wird, muss man in der Menüleiste unter Ansicht auf Toolbox klicken, dann wird es im linken Bereich der Arbeitsoberfläche angezeigt. Es befinden sich folgende zusätzliche Registerkarten in der Toolbox, siehe auch Abbildung 9.5:

- SIMATICIT WebControls
- Production Suite Library
- CAB Providers
- Messaging Manager
- CAB Library

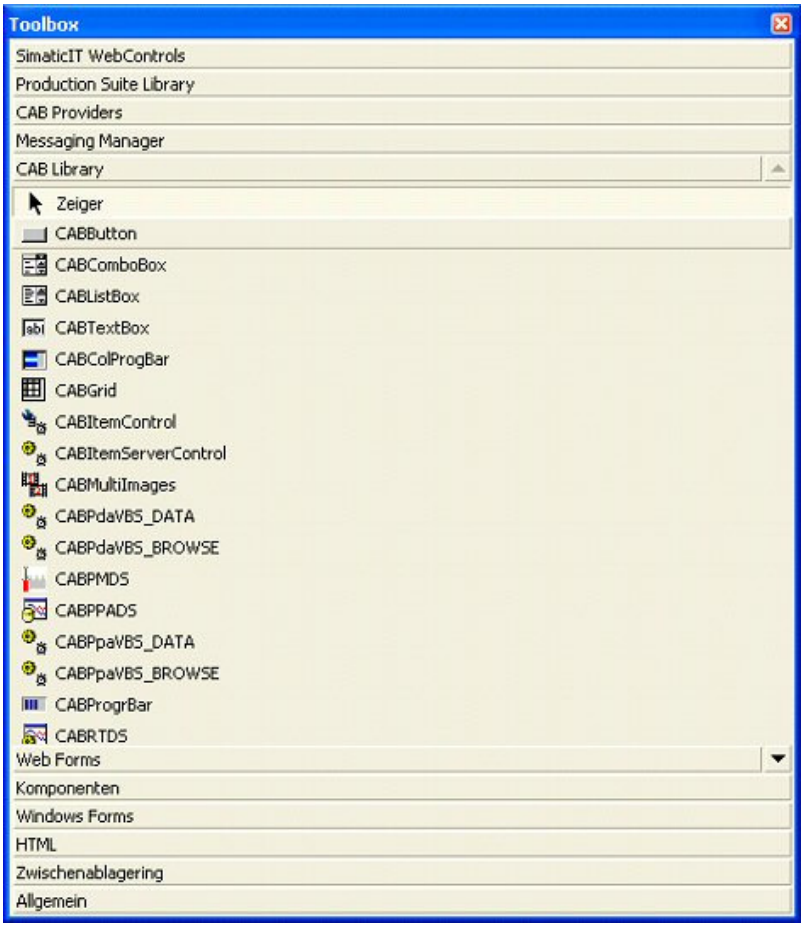

Abbildung 9.5: Toolbox

Die Registerkarte „CAB Library“ enthält die wichtigsten CAB-Controls (z.B. CAB-Button) um das MES-Projekt zu steuern. Die Erklärung der einzelnen eingesetzten Elemente folgt im nächsten Kapitel und zwar bei der Beschreibung der einzelnen Seiten der Web-Anwendung „DemoCAB“. Alle anderen Elemente wie Label, Data-Grid oder Text-Box usw. wurden aus der Karteikarte „Web Forms“ entnommen und sind Standardelemente von MS Visual Basic.NET. Erklärungen zu diesen Elementen findet man in der Menüleiste unter Hilfe sowie in einschlägiger Fachliteratur.

9.4 Beschreibung der Web-Anwendung „DemoCAB“

Die Web-Anwendung zur Steuerung des MES-Projekts wurde „DemoCAB“ genannt und besteht aus mehreren Web-Formularen, welche in das Projekt einfügt wurden. Zu diesem Zweck klickt man im Projektmappen Explorer mit der rechten Maustaste auf die Web-Anwendung (in diesem Fall auf WebApplication1) und anschließend auf Hinzufügen – Web-Form hinzufügen, siehe Abbildung 9.6.

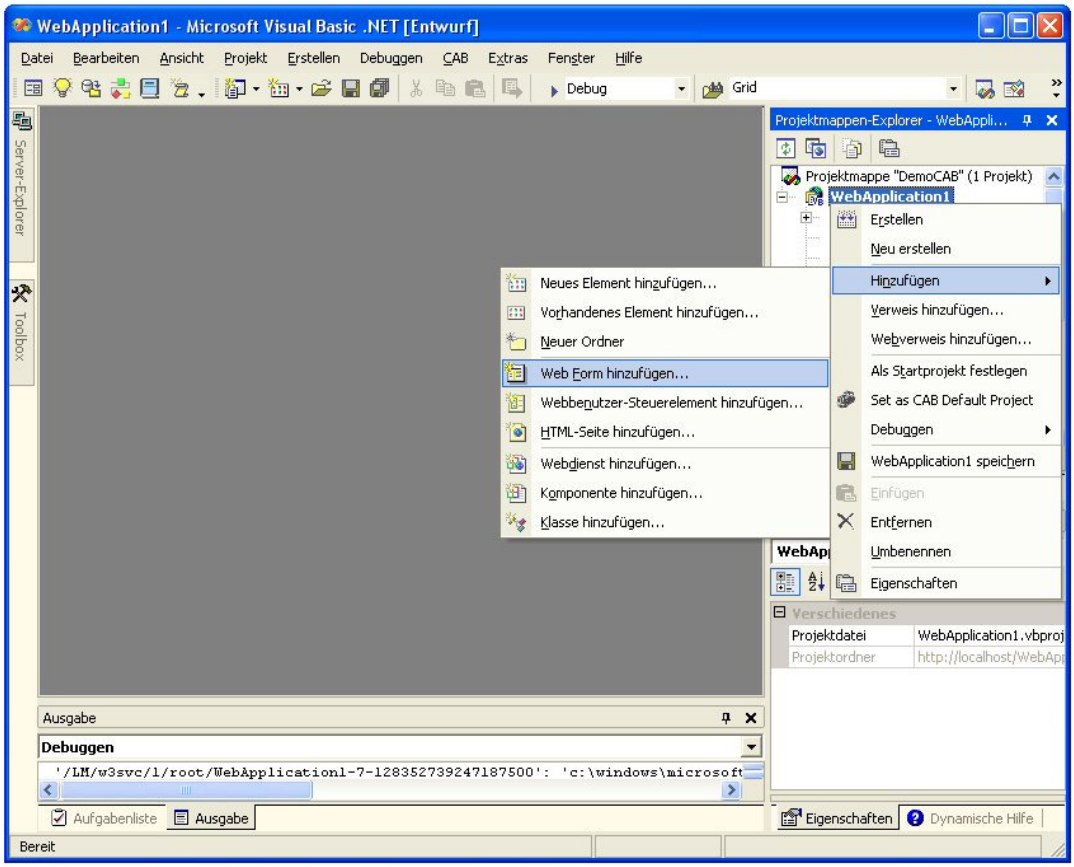

Abbildung 9.6: Ein Web-Formular zu einer Web-Anwendung hinzufügen

Anschließend öffnet sich das Fenster „Neues Element hinzufügen“, wie in Abbildung 9.7 dargestellt. Unter Vorlagen wählt man ein „Web Form“ aus und im Eingabefeld Name wird die dafür vorgesehene Bezeichnung festgelegt.

Die Endung „.aspx“ steht für ASP.NET und deutet darauf hin, dass das hinzugefügte Element ein Web-Formular ist. Im Projektmappen-Explorer (befindet sich auf der rechten Seite im Hauptfenster) wird nun das eingefügte Web-Formular angezeigt. Es ist von Vorteil gleich eine Namensänderung vorzunehmen bevor man mit der Gestaltung der Seite beginnt, da spätere Änderungen der Namen immer ein Risiko für Programmierfehler darstellen. Ähnliches gilt für eingesetzte Elemente wie Button, TextBox, DataGrid usw., hierfür sollte die Namensgebung schon auf die Verwendung hinweisen um die Programmierung für Dritte leichter verständlich zu machen.

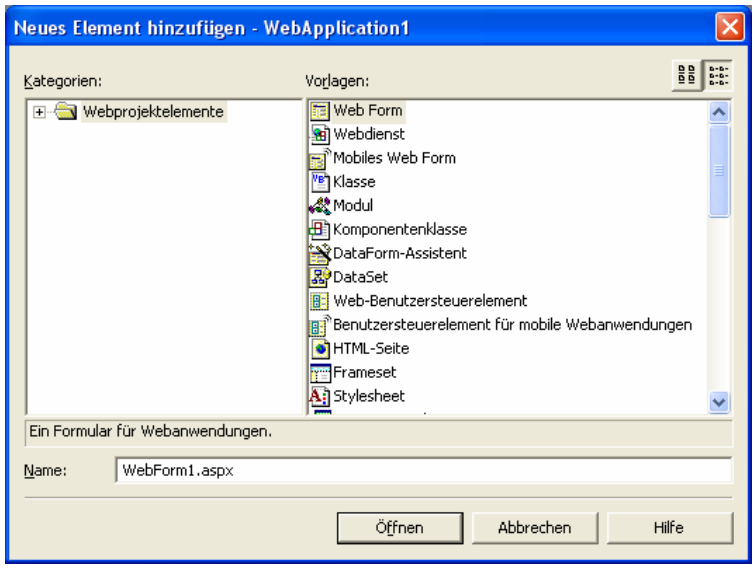

Abbildung 9.7: Ein Web Form hinzufügen

9.4.1 Das „Login“ Web-Formular

Die Startseite unserer Anwendung ist eine übliche Login-Aufforderung, um die Anlage vor nicht berechtigtem Gebrauch zu schützen und heißt „Login.aspx“. Diese Seite soll als Startseite angezeigt werden und erst nach erfolgreicher Anmeldung soll der Anwender auf die Hauptseite der Web-Anwendung gelangen. Die dafür notwendige Einstellung ist sehr einfach zu treffen. Durch einen Klick mit der rechten Maustaste auf das gewünschte Web-Formular (in unserem Fall auf Login.aspx) und anschließendes Klicken auf „Als Startseite Festlegen“ ist die richtige Einstellung festgelegt. Sie kann relativ einfach wieder verändert werden, siehe Abbildung 9.8.

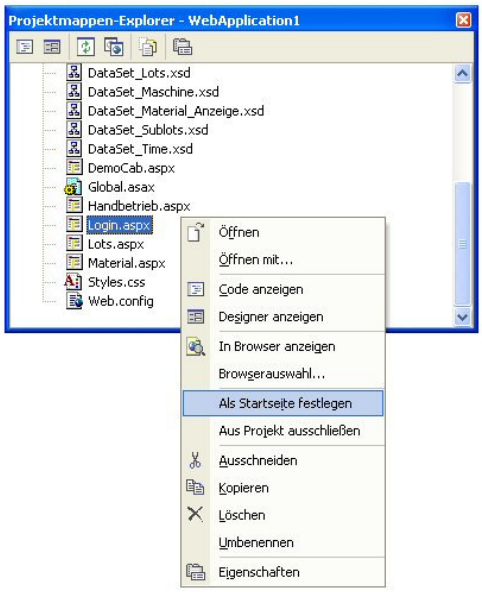

Abbildung 9.8: Login.aspx als Startseite festlegen

Die Erstellung von Usern und deren Passwörtern geschieht in der Management Konsole von SIMATIC IT Production Suite und zwar in der Menüleiste unter Tools – User Manager, siehe Abbildung 9.9.

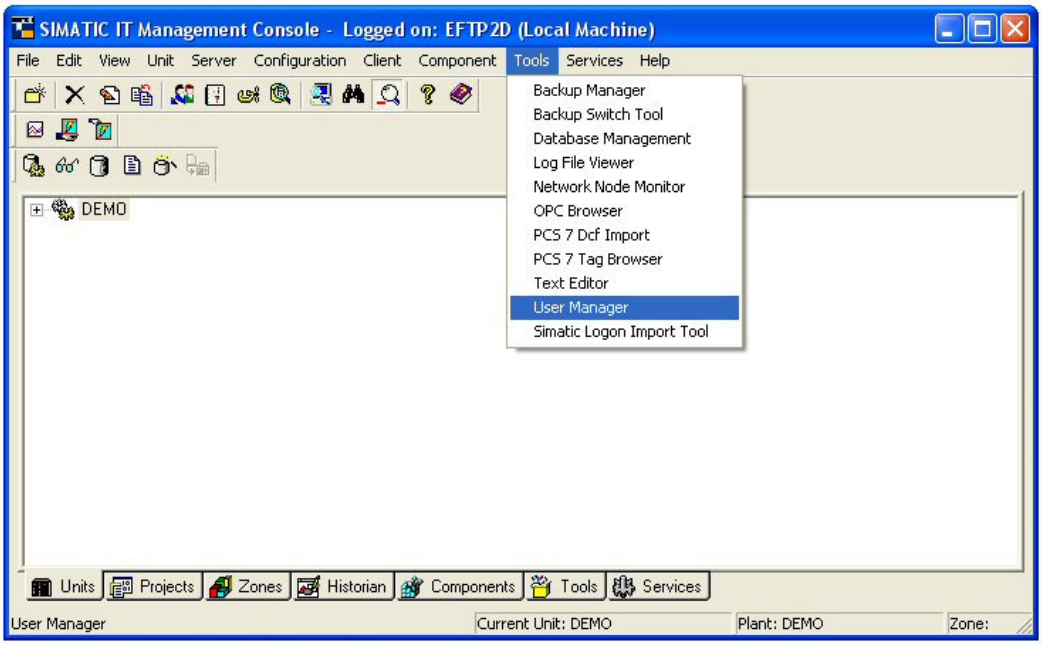

Abbildung 9.9: User Manager

Hier können auch unterschiedliche Rechte der Benutzer vergeben werden, ähnlich der Windows-Benutzerkonten-Verwaltung.
CAB stellt für die Benutzerabfrage ein sehr einfaches Element zur Verfügung. In der Toolbox befindet sich unter der Registerkarte „CAB Providers“ ein Element mit dem Namen „CABLoginWebControl“, mit welchem auf sehr schnelle Art und Weise eine Login Abfrage abgebildet werden kann. Um ein Element aus der Toolbox in das Web-Formular einzufügen, muss man es mit gehaltener Maustaste in das Formular ziehen. Diese Art der Handhabung nennt man „Drag and Drop“ (Ziehen und Fallenlassen) und ist sehr gebräuchlich sowie benutzerfreundlich.
Wie in Abbildung 9.10 dargestellt, braucht man für einen Login-Aufruf noch zwei Text-Box-Elemente, zwei Labels und einen Button. Alle drei Elemente findet man in der Toolbox unter der Registerkarte „Web Forms“. Ein Label ist ein Element ohne besondere Funktionen und wird dazu verwendet Texte anzuzeigen, welche nicht verändert werden müssen. In unserem Beispiel dienen sie zur Anzeige der beiden Eingabeaufforderungen „User Name:“ und „Password:“.

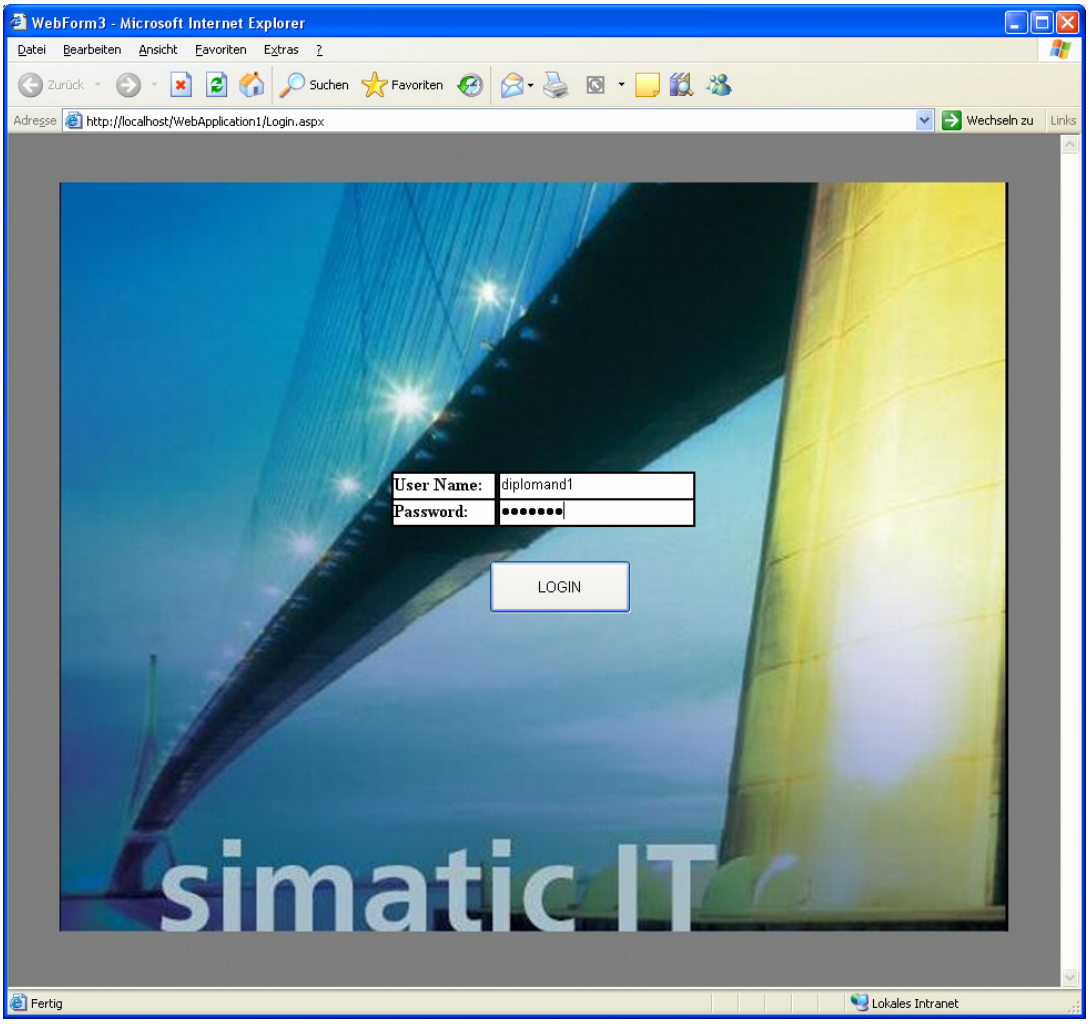

Abbildung 9.10: „Login“-Web-Formular

Auf der rechten Seite des MS Visual Studio Hauptfensters sollte das Eigenschaften-Fenster angezeigt werden, in welchem alle möglichen Eigenschaften eines Elements verändert werden können, siehe Abbildung 9.11. Wenn dies nicht der Fall ist, kann man in der Menüleiste unter Ansicht auf Eigenschaften-Fenster klicken und dieses wird dann angezeigt. Weiters kann man mit der rechten Maustaste auf ein Objekt klicken und dann Eigenschaften wählen, um das Eigenschaften-Fenster zu öffnen. Die meisten Eigenschaften dienen zur Einstellung von Größe, Farbe, Rahmenanzeige usw. und werden hier nicht näher erläutert. Wichtig ist die Eigenschaft „ID“, da das Objekt mit diesem Namen im Programmiercode angesprochen wird. Weiters ist die Eigenschaft „Text“ von Bedeutung, hier kann der Inhalt, der angezeigt werden soll, hinterlegt werden. Bei Labels ist die ID-Eigenschaft nicht so wichtig, da keine Änderungen des Objekts im Programmcode vorgesehen sind. Deshalb wurden in dieser Anwendung die beiden Labels nicht umbenannt und die ID-Einstellung bei „Label1“ und „Label2“ belassen, siehe Abbildung 9.11. Ähnliches gilt auch für einen Button und eine TextBox.

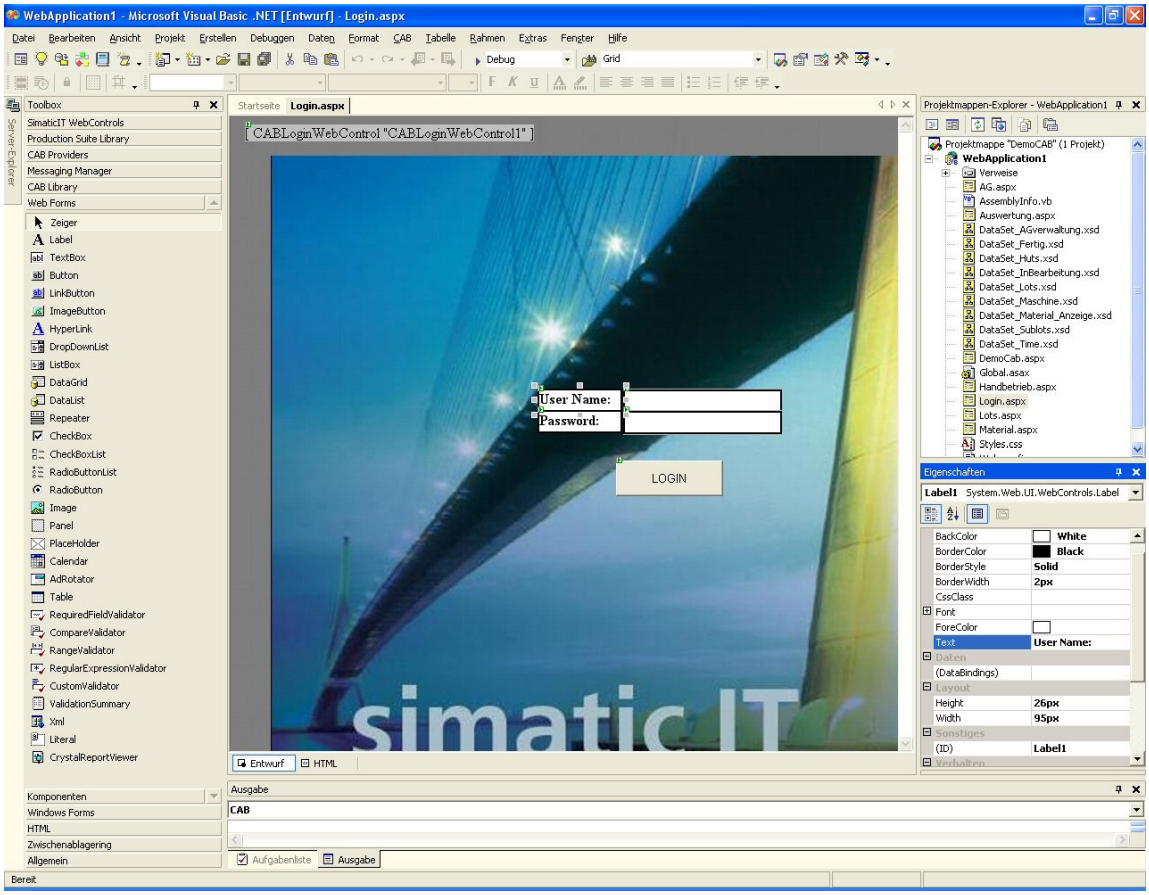

Abbildung 9.11: Eigenschaften-Fenster in MS Visual Basic.NET

Bei der Eingabe eines Passwortes möchte man nach Möglichkeit die eingegebenen Zeichen nicht wirklich anzeigen sondern durch das Zeichen „*“ ersetzen. Diese Einstellung ist ebenfalls leicht zu realisieren. Unsere beiden Text-Box-Elemente heißen „User1“ und „Pass1“. Wie der Name schon verrät, soll im Textfeld „Pass1“ das Passwort eingegeben werden. Die Eigenschaft, die hierfür verändert wird, heißt „TextMode“ und hat drei Möglichkeiten zur Auswahl:

- SingleLine
- MultiLine und
- Password.

Die Standardeinstellung ist „SingleLine“ und ermöglicht eine einzeilige Eingabe, „MultiLine“ eine mehrzeilige, siehe Abbildung 9.12 rechts unten. Bei der Einstellung „Password“ werden alle eingegebenen Symbole durch das Zeichen „*“ ersetzt und damit wird eine Passworteingabe ermöglicht, wie in Abbildung 9.10 dargestellt.

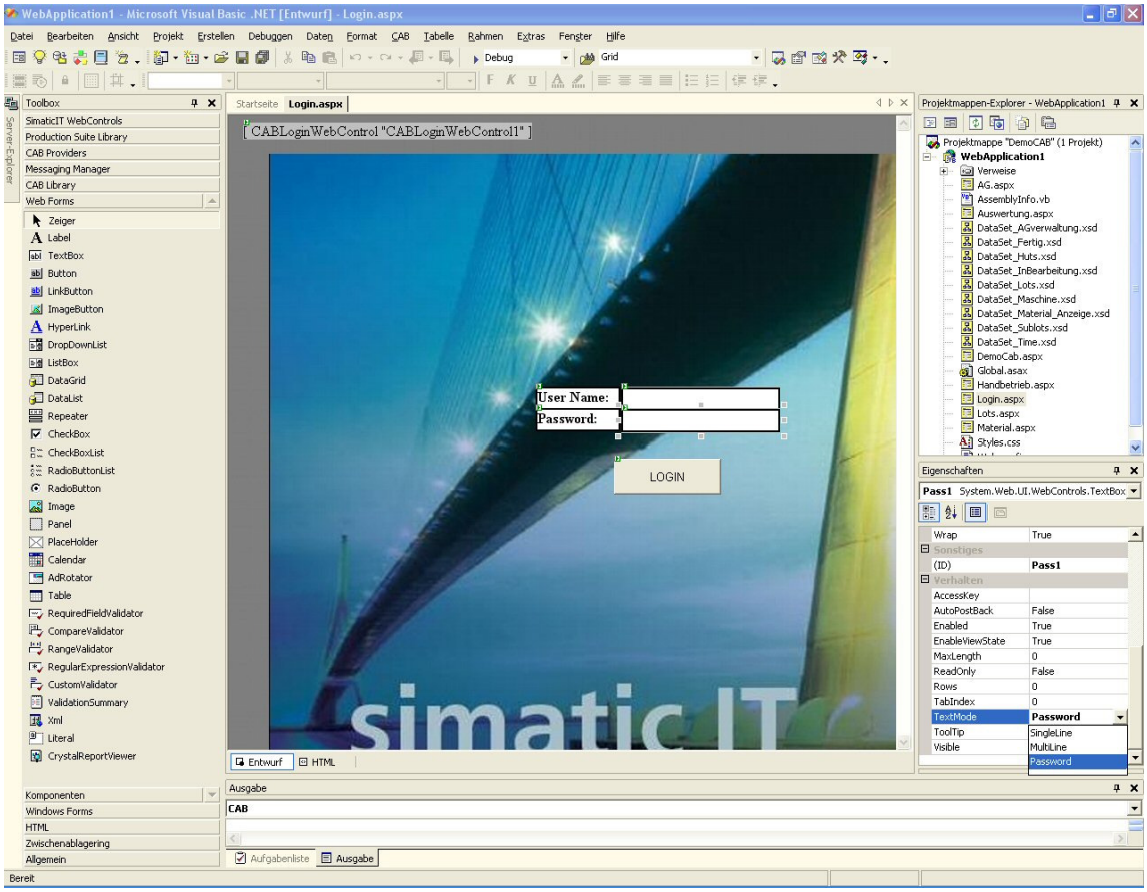

Abbildung 9.12: Passworteingabe mit Hilfe einer „TextBox“

Mit einem Klick auf den Login-Button soll der Anmeldevorgang abgeschlossen werden und man gelangt bei korrekter Eingabe in das Hauptmenü, welches im nächsten Web-Formular beschrieben wird, siehe Kapitel 9.4.2.
Wie zu Beginn dieses Kapitels erläutert benötigt man nur ein „CabLoginWebControl-Element“, welches die Login-Daten überprüft und mit den Einstellungen des User Managers von SIMATIC IT Production Suite vergleicht. Den Programmcode hierfür muss man nicht mehr schreiben, lediglich eine Übergabe der beiden Textfelder ist noch durchzuführen. Bei falscher Eingabe wird das Login Fenster erneut geladen und das Eingabefeld für das Passwort ist wieder leer. Die Übergabe der Textfelder erfolgt durch klicken auf den Login Button. Das dadurch ausgelöste Ereignis „Login_Click“ und der dazugehörige Programmcode sehen wie folgt aus:

```
Private Sub Login_Click(ByVal sender As System.Object, _
ByVal e As System.EventArgs) Handles Login.Click
   CABLoginWebControl1.Login(User1.Text, Pass1.Text)
End Sub
```

Um in MS Visual Basic in die Programmiercodeebene zu gelangen gibt es mehrere Möglichkeiten. Der einfachste Weg ist ein rechter Mausklick irgendwo im Web-Formular und anschließend auf „Code anzeigen“ klicken. Um gleich direkt in das Unterprogramm „Private Sub Login_Click“ zu gelangen genügt ein Doppelklick auf den Login-Button. „Login“ ist die ID des Buttons, deshalb heißt das Unterprogramm auch „Login_Click“.

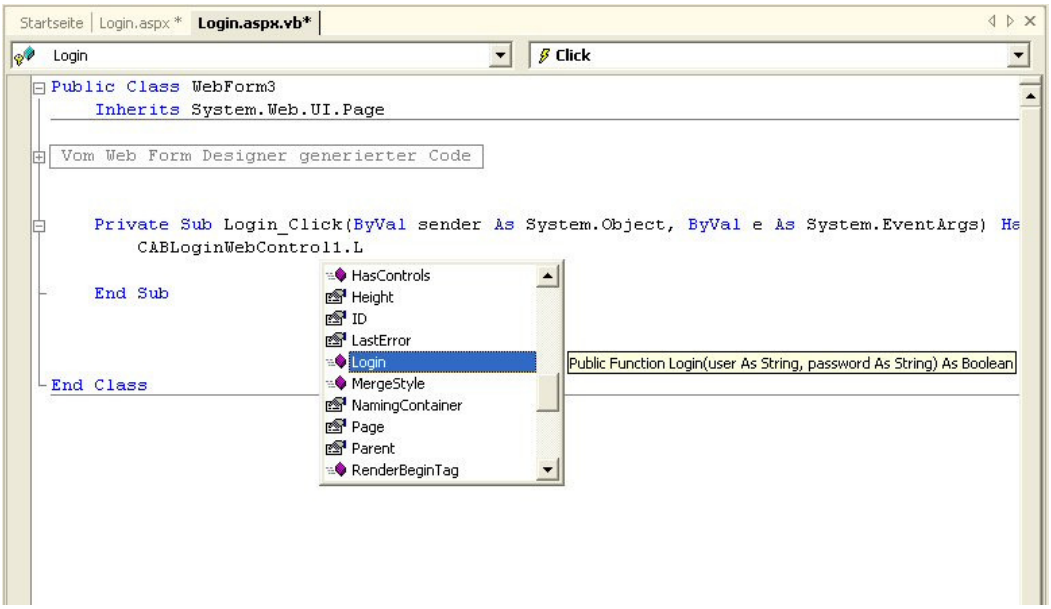

Abbildung 9.13: Programmcode für das „Login“-Web-Formular

Das eingefügte „CabLoginWebControl“-Element hat die Identität „CABLoginWebControl1“ und wird im Programmcode unter dieser Bezeichnung eindeutig angesprochen. Um die Eigenschaften und Unterprogramme (als auch Funktionen) dieses Objekts zu sehen, muss man nach dem Namen einen Punkt eingeben. Danach wird eine automatische Auswahlliste angezeigt, aus welcher man mit Hilfe der Pfeiltasten eine Auswahl treffen kann, siehe Abbildung 9.13. Die Funktion „Login“ wird für unseren Zweck benötigt und sie fordert zwei Übergabeparameter als „String“ Variablen:

- user und
- password.

Die beiden String-Variablen werden vom Anwender in die beiden Textfelder „User1“ und „Pass1“ eingegeben und können mit der Eigenschaft „User1.Text“ sowie „Pass1.Text“ angesprochen werden. Deshalb sieht die einzige Zeile die in das Unterprogramm eingefügt werden muss wie folgt aus:

```
CABLoginWebControl1.Login(User1.Text, Pass1.Text)
```

Man muss der Web-Anwendung noch mitteilen, welche Seite (Web-Formular) bei positiver Anmeldung angezeigt werden soll. Diese Einstellung erfolgt in der Datei „Web.config“, welche man im Projektmappen-Explorer findet. Durch Doppelklicken auf den Namen öffnet sich ein Fenster mit dem dazugehörigen HTML-Code, siehe Abbildung 9.14.

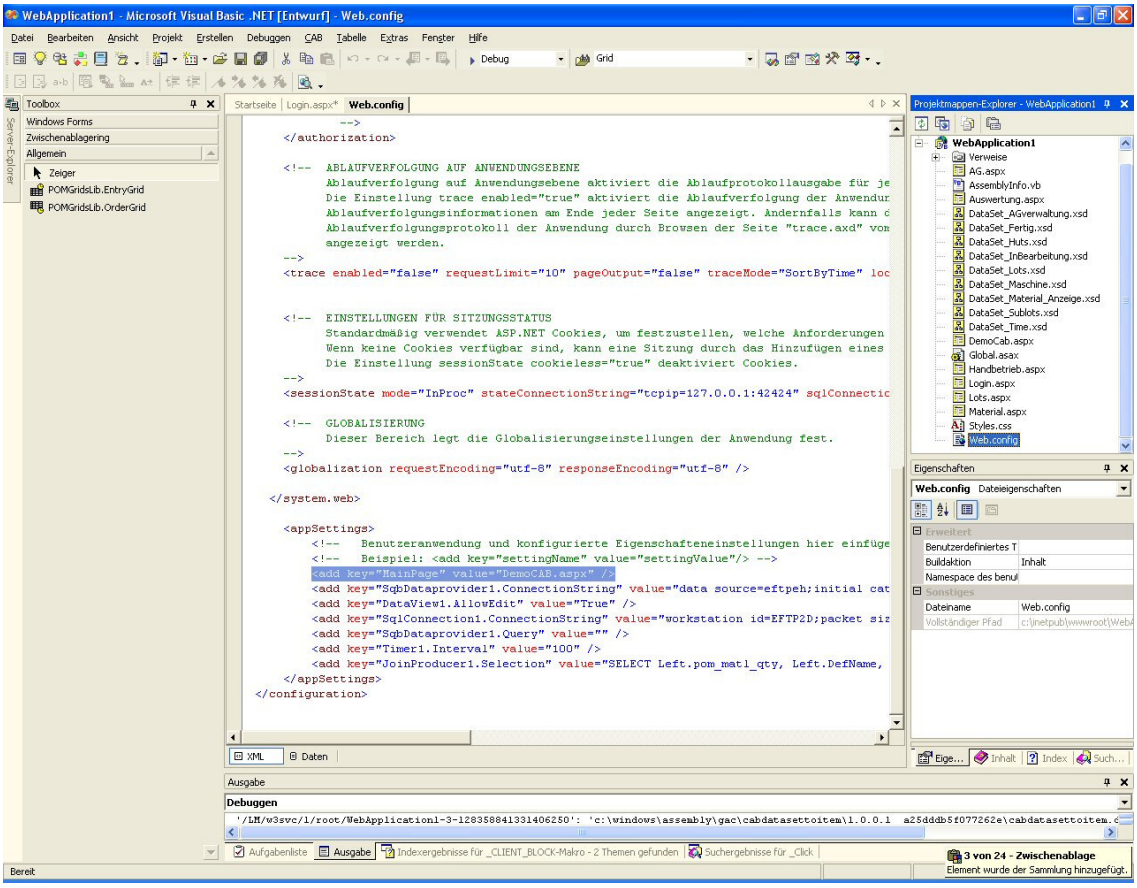

Abbildung 9.14: „Web.config“-Datei (HTML Code)

Die in Abbildung 9.14 markierte Textstelle muss erst manuell eingefügt werden, alle anderen Zeilen wurden automatisch generiert. Man legt hierbei fest, dass „DemoCAB.aspx“ die Hauptseite unserer Web-Anwendung ist und diese wird nach erfolgreichem Login auch aufgerufen. Der genaue Programmcode lautet:

```
<add key="MainPage" value="DemoCAB.aspx" />
```

Genauere Informationen zu HTML-Programmierung ist einschlägiger Fachliteratur zu entnehmen.

9.4.2 Hauptmenü

Nach erfolgreicher Anmeldung öffnet sich also automatisch das „DemoCAB.aspx“-Web-Formular, welche als Hauptmenü der ganzen Web Anwendung dient. Auf der rechten Seite befindet sich immer das Menü, wo man verschiedene Buttons zur Auswahl hat, siehe Abbildung 9.15. Dieses Schema wird auch auf allen weiteren Seiten beibehalten.

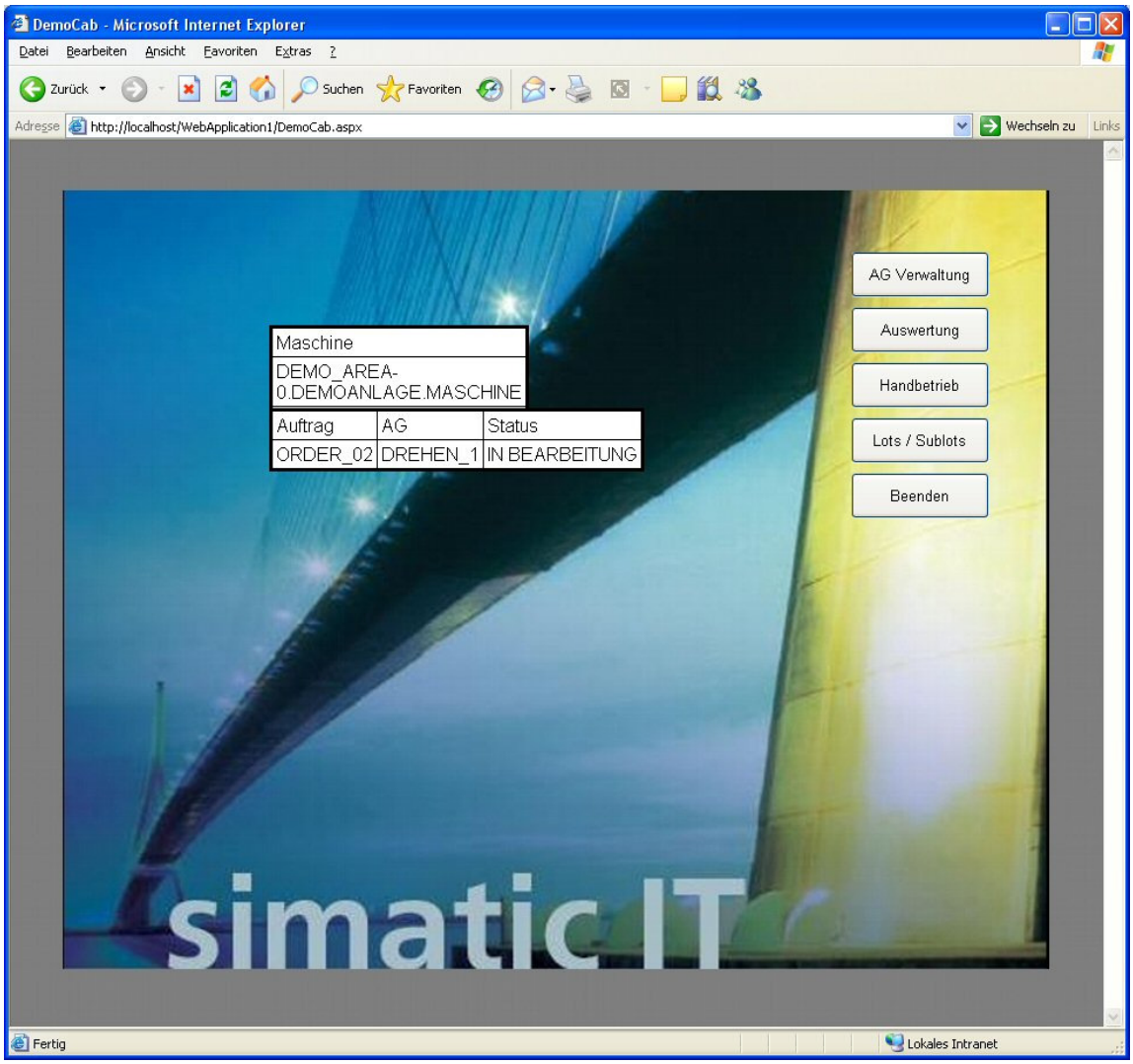

Abbildung 9.15: Hauptmenü der Web-Anwendung „Demo CAB“

In der Mitte wird die Maschine angezeigt und der Arbeitsgang eines Auftrags, welcher zurzeit an der Maschine abgearbeitet wird. Hierfür werden in MS Visual Basic so genannte „DataGrids“ verwendet. Mit deren Hilfe kann man Inhalte aus einer Datenbank anzeigen. Man findet sie in der Registerkarte „Web Forms“.

Unsere Bearbeitungsmaschine heißt „Maschine“, diese Bezeichnung wird von der Datenbank übernommen und wird im Production Modeler vergeben. Im Menü auf der rechten Seite stehen folgende Buttons zur Verfügung:

- AG Verwaltung
- Auswertung
- Handbetrieb
- Lots/Sublots
- Beenden

Wenn man auf einen Button klickt soll eine neue Webseite aufgerufen werden. Eine Ausnahme stellt der Button „Beenden“ dar, welcher die Anwendung schließt.
Der Programmcode sieht folgendermaßen aus:

```
Private Sub LotsBtn_Click(ByVal sender As System.Object, _
ByVal e As System.EventArgs) Handles LotsBtn.Click
    Response.Redirect("Lots.aspx")
End Sub
Private Sub HandbetriebBtn_Click(ByVal sender As System.Object, _
ByVal e As System.EventArgs) Handles HandbetriebBtn.Click
    Response.Redirect("Handbetrieb.aspx")
End Sub
Private Sub ArbeitsgangBtn_Click(ByVal sender As System.Object, _
ByVal e As System.EventArgs) Handles ArebitsgangBtn.Click
    Response.Redirect("AG.aspx")
End Sub
Private Sub AuswertungBtn_Click(ByVal sender As System.Object, _
ByVal e As System.EventArgs) Handles AuswertungBtn.Click
    Response.Redirect("Auswertung.aspx")
End Sub
Private Sub BeendenBtn_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles BeendenBtn.Click
    RegisterClientScriptBlock("DemoCab.aspx", _
"<Script>close ()</Script>")
End Sub
```

Um beispielsweise auf die Seite „Lots.aspx“ zu wechseln, muss der Benutzer den Button „LotsBtn“ anklicken und dadurch wird das Unterprogramm „Private Sub LotsBtn_Click“ durchlaufen. Die einzige Zeile die manuell eingetragen werden muss lautet:

```
Response.Redirect("Lots.aspx")
```

Dadurch wird das Web-Formular „Lots.aspx“ geladen und im Webbrowser angezeigt.

Um Daten aus einer Datenbank auszulesen verwendet man SQL (Structured Query Language), was zu Deutsch nichts anderes heißt als strukturierte Abfragesprache. SQL ist eine standardisierte Sprache von ANSI als auch von ISO und wird von fast allen gängigen Datenbanksystemen unterstützt.

Um eine SQL-Abfrage (SQL-Query) zu tätigen, stellt CAB ein sehr einfaches Tool zur Verfügung. Ein „SQBDataprovider“, welcher sich in der Registerkarte „CAB Providers“ befindet, wird hierfür benötigt. SQB (Smart Query Builder) ist ein Zusatzprogramm von SIMATIC IT um SQL-Abfragen zu generieren und als SQL-Dateien zu speichern. Man kann allerdings nur auf die Datenbank „SITMesDb“ von SIMATIC IT Production Suite zugreifen, welche bei der Installation automatisch erstellt worden ist. Man kann in MS Visual Basic auch direkt über den Programmiercode auf Datenbanken zugreifen, dies wird aber hierfür nicht benötigt. Um eine SQL-Datei zu erstellen braucht man jedoch nicht zwangsläufig den Smart Query Builder, es reicht ein einfacher Texteditor völlig aus, um die Abfrage manuell zu schreiben. Für dieses Web-Formular wurden zwei „SQBDataprovider“ verwendet, siehe Abbildung 9.16.

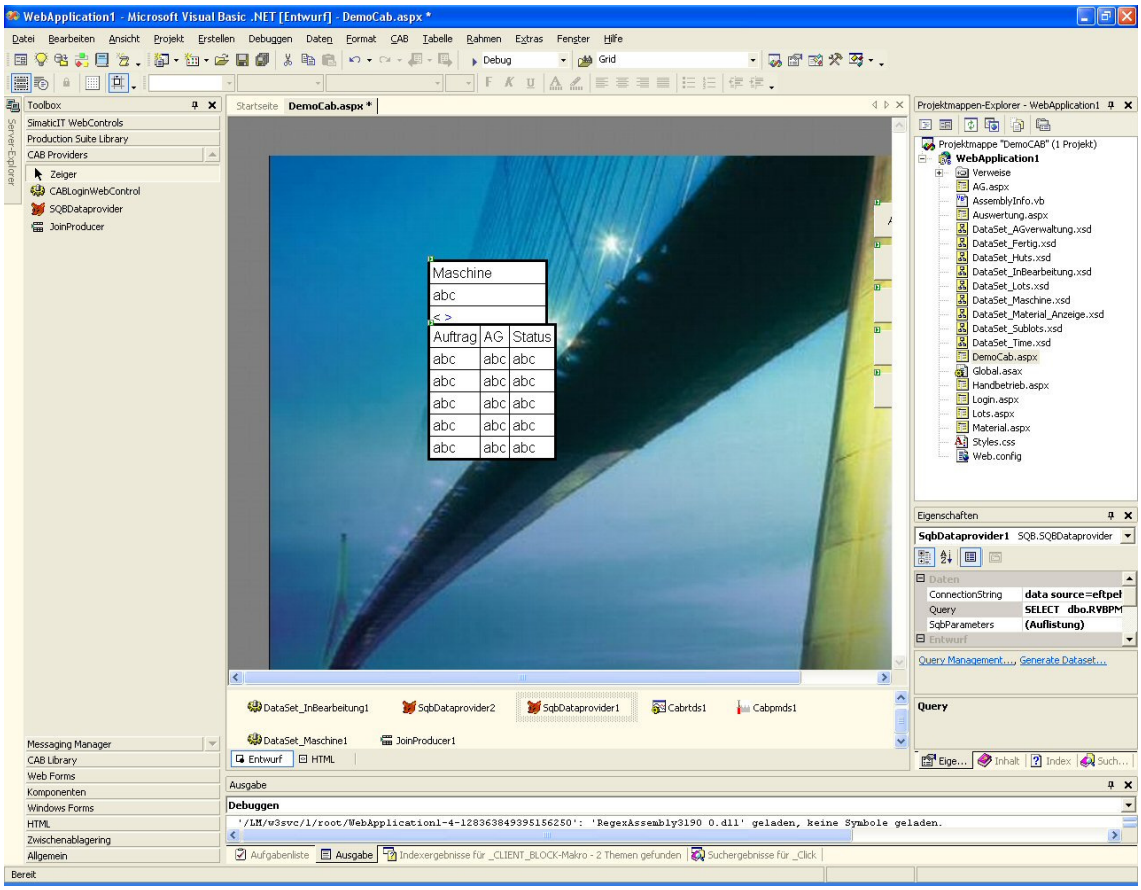

Abbildung 9.16: „SQBDataprovider“ in MS Visual Basic

Im Fenster Eigenschaften eines „SQBDataprovider“ gibt man unter „ConnectionString“ folgende Daten ein:

```
data source=eftpeh;initial catalog=SITMesDb;user -
id=sa;password=*******
```

Damit weiß der „SQBDataprovider“ welche Datenquelle, in unserem Fall ein Zweitrechner mit dem Computernamen „eftpeh“ auf dem der SQL Server läuft, welche Datenbank (SITMesDb) und die nötigen Anmeldedaten wie „user id“ und „password“. Die nächste Eigenschaft heißt „Query“ und kann entweder direkt als SQL-Code eingegeben werden oder man verweist auf eine SQL-Datei in der die Abfrage steht. Dazu muss die SQL-Datei aber in ein bestimmtes Verzeichnis gespeichert werden („C:\ICUBESYS\SIT\SQB“). Erstellt man eine SQL-Abfrage mit dem Smart Query Builder wird sie automatisch dort gespeichert. „C:\ICUBESYS\SIT“ ist das Installationsverzeichnis von SIMATIC IT Production Suite. Durch Drücken auf den Hyperlink „Query Management“ (unter dem Fenster Eigenschaften) wird der Smart Query Builder geöffnet. Die Erklärung dieses Programms entnehmen sie der Hilfedatei. Die SQL-Abfrage für das „DataGrid“ zur Anzeige des gerade in Bearbeitung stehenden Arbeitsganges sieht folgendermaßen aus:

```
SELECT
  dbo.RVPOM_ORDER.pom_order_id,
  dbo.RVPOM_ENTRY.pom_entry_id,
  dbo.RVPOM_ENTRY_STATUS.id
FROM
  dbo.RVMM_MATERIAL_INFO  MaterialForOrder RIGHT OUTER JOIN
dbo.RVPOM_ORDER ON
(MaterialForOrder.DefID=dbo.RVPOM_ORDER.matl_def_id)
   LEFT OUTER JOIN dbo.RVPOM_ENTRY ON
(dbo.RVPOM_ORDER.pom_order_pk=dbo.RVPOM_ENTRY.pom_order_pk)
   LEFT OUTER JOIN dbo.RVPOM_ENTRY_STATUS ON
(dbo.RVPOM_ENTRY.pom_entry_status_pk=dbo.RVPOM_ENTRY_STATUS.pom_entry_
status_pk)

WHERE
(dbo.RVPOM_ENTRY_STATUS.id = 'IN BEARBEITUNG')
```

Mit dieser SQL-Abfrage werden drei Einträge ausgelesen, welche in der „SITMesDb“ Datenbank gespeichert sind:

- Order ID,
- Entry ID und
- Entry Status.

Um diese Daten jetzt in einem „DataGrid“ anzuzeigen benötigt man noch ein so genanntes „DataSet“, also einen Datensatz, in dem die ausgelesenen Daten gespeichert werden. Ein solches „DataSet“ erhält man durch Anklicken des Hyperlink „Generate Dataset…“, welcher sich unter dem Eigenschaften-Fenster befindet, siehe Abbildung 9.16. Danach wird eine Eingabemaske angezeigt, in der man den Namen des neuen „DataSets“ eingeben muss. Nach Drücken der „OK“-Taste ist der neue Datensatz auch schon erstellt. Im Programmcode sind noch folgende Einträge manuell zu tätigen, um die Daten im gewünschten „DataGrid“ anzuzeigen:

```
Private Sub Page_Load(ByVal sender As System.Object, _
ByVal e As System.EventArgs) Handles MyBase.Load
    SqbDataprovider1.Fill(DataSet_Maschine1)
    DataGrid1.DataBind()
    SqbDataprovider2.Fill(DataSet_InBearbeitung1)
```

```
    DataGrid2.DataBind()
End Sub
```

Das Unterprogramm „Private Sub Page_Load“ wird immer ausgeführt wenn die Seite neu geladen wird. Es bietet sich also an, die oben angeführten Programmzeilen hier zu platzieren. Die Zeile „SqbDataprovider1.Fill(DataSet_Maschine1)“ füllt den Datensatz „DataSet_Maschine1“ mit den neuen Werten aus der SQL-Datenbank. Die Zeile „DataGrid1.DataBind()“ speichert (bindet) diese Werte im „DataGrid1“. Die Zuordnung vom „DataSet_Maschine1“ zum „DataGrid1“ erfolgt über die Eigenschaften des „DataGrids“, siehe Abbildung 9.17.

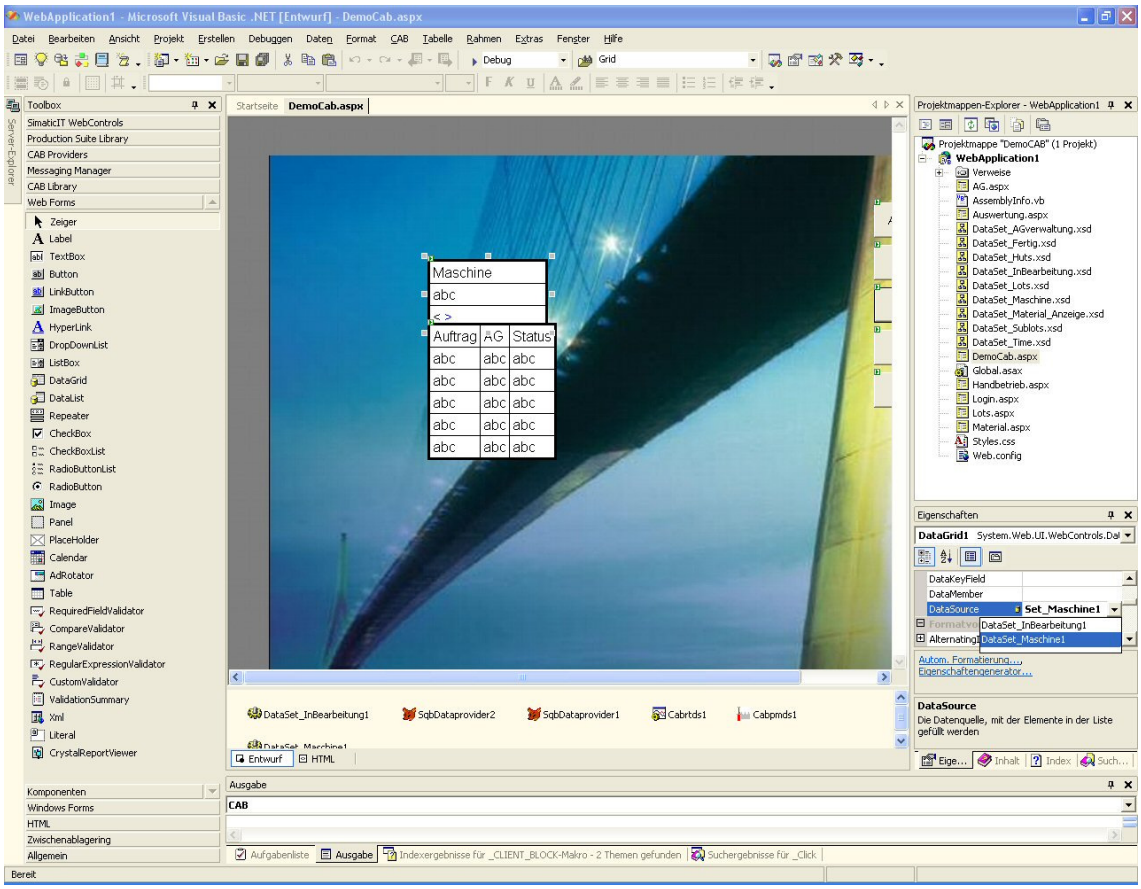

Abbildung 9.17: „DataSource“-Eigenschaft in einem „DataGrid“

In der Eigenschaft „DataSource“ kann man bestehende „DataSets“ auswählen und so dem „DataGrid“ zuordnen. Unter dem Fenster „Eigenschaften“ befindet sich dazu ein Hyperlink zum Eigenschaftengenerator des „DataGrids“. Mit einem rechten Mausklick auf das „DataGrid“ kommt man ebenfalls zu diesem Generator. In Abbildung 9.18 kann man erkennen, welche Möglichkeiten er zum Design eines „DataGrids“ bietet.

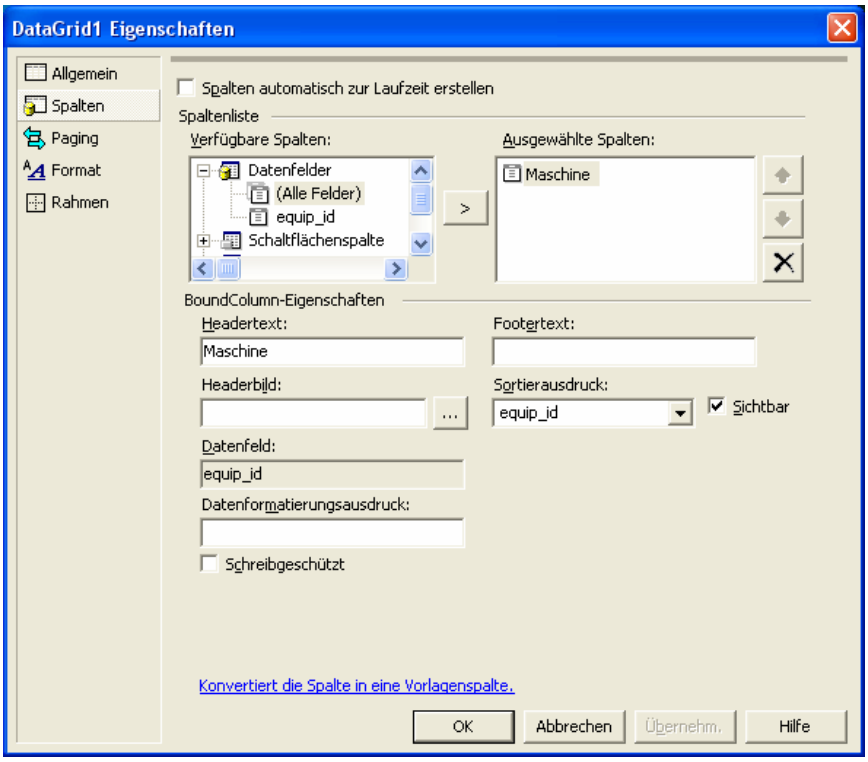

Abbildung 9.18: Eigenschaftengenerator eines „DataGrid“-Elements

Neben allgemeinen Einstellungen, wie Format und Rahmen, kann man auch Spalteneigenschaften verändern oder hinzufügen (z.B. Einfügen einer Schaltflächenspalte), siehe Kapitel 9.4.3. Um nicht unendlich viele Einträge auf einmal anzuzeigen, kann man sich die Einstellung „Paging“ zu nutze machen und nur eine bestimmte Anzahl von Zeilen pro Seite festlegen. Durch einen Mausklick kann der Anwender dann zur nächsten oder auch zur vorigen Seite gelangen.

Weitere Möglichkeiten, um in die Anlage einzugreifen, sind im Hauptmenü nicht vorgesehen. Es soll nur einen Überblick geben, welcher Arbeitsgang auf der Maschine gerade abgearbeitet wird. Weiters soll eine Menüsteuerung, ähnlich einer CNC-Steuerung, vorhanden sein.

9.4.3 Das Web-Formular „Arbeitsgang Verwaltung“

Das nächste Web-Formular ist zur Steuerung der Anlage gedacht und trägt den Namen „AG.aspx“. Hier werden alle freigegebenen Arbeitsgänge angezeigt, die auf dieser Maschine abgearbeitet werden sollen. Dazu wurde ein „DataGrid“ verwendet, welches folgende aus einer SQL-Abfrage generierte Daten anzeigt:

- Auftrag
- Arbeitsgang
- Menge
- Produkt

- Status des Arbeitsgangs

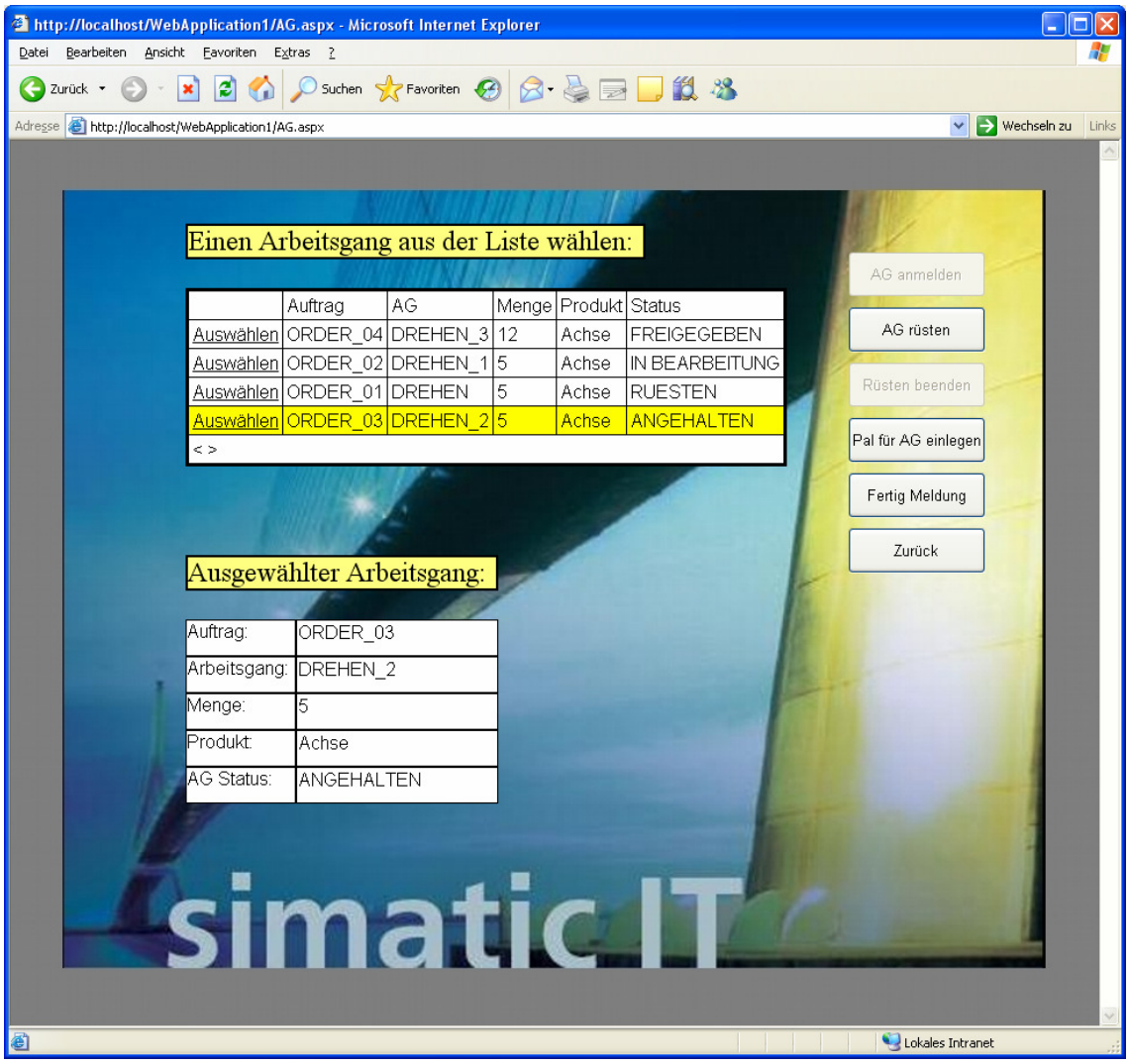

Abbildung 9.19: Web-Formular „AG-Verwaltung“

Auf der linken Seite des oberen „DataGrids“ wurde mit Hilfe des Eigenschaftengenerators eine „Schaltflächenspalte“ hinzugefügt, welche zur Auswahl eines Arbeitsgangs verwendet wird. Im unteren Bereich dieser Seite wird der ausgewählte Arbeitsgang nochmals in Zeilenform angezeigt. Hierfür wurden fünf Text-Box-Elemente verwendet. Auf der rechten Seite des Web-Formulars wird das übliche Menü, wie auch schon im Web-Formular „Hauptmenü“, verwendet. Je nach dem welcher Status für den Arbeitsgang gerade hinterlegt ist, werden verschiedene Buttons im Menü aktiv oder inaktiv. Dies wird über die Button-Eigenschaft „Enabled“ gesteuert. Folgende Buttons stehen im Menü zur Verfügung:

- AG anmelden
- AG rüsten
- Rüsten beenden

- Palette für AG einlegen (CAB Button)
- Fertigmeldung
- Zurück

Welche Statuswechsel möglich sind wird in Kapitel 8.1.2 beschrieben. Der Button „Palette für AG einlegen“ soll den Automatikbetrieb für eine Palette auslösen. Nach dem Einlegen der Palette werden alle Verfahrbewegungen automatisch durchlaufen. Die Bearbeitung wird durchgeführt und anschließend wird die Palette wieder zur Be-/Entladestation transportiert und entnommen. Alle diese Schritte können im „Handbetrieb“ auch einzeln durchlaufen werden, siehe Kapitel 9.4.4.

Der Programmiercode für die Seite „AG-Verwaltung“ ist im folgenden Teil beschrieben. Zeilen die mit einem „'“ beginnen sind Kommentarzeilen und werden für die Erklärung verwendet.

```
Private Sub Page_Load(ByVal sender As System.Object, _
ByVal e As System.EventArgs) Handles MyBase.Load
    SqbDataprovider1.Fill(DataSet_AGverwaltung1)
    DataGrid1.DataBind()
End Sub
```

Das Ereignis „Page_Load“ wird aufgerufen, wenn die Seite neu geladen wird. Die beiden Programmzeilen sind zum Füllen des Datensatzes und zum Speichern im „DataGrid“, siehe Kapitel 9.4.2. Die SQL-Abfrage für den Datensatz „DataSet_AGverwaltung1“ sieht folgendermaßen aus:

```
SELECT
  dbo.RVPOM_ORDER.pom_order_id,
  dbo.RVPOM_ENTRY.pom_entry_id,
  dbo.RVPOM_ORDER.pom_matl_qty,
  MaterialForOrder.DefName,
  dbo.RVPOM_ENTRY_STATUS.id
FROM
  dbo.RVMM_MATERIAL_INFO MaterialForOrder RIGHT OUTER JOIN
dbo.RVPOM_ORDER ON
(MaterialForOrder.DefID=dbo.RVPOM_ORDER.matl_def_id)
  LEFT OUTER JOIN dbo.RVPOM_ENTRY ON
(dbo.RVPOM_ORDER.pom_order_pk=dbo.RVPOM_ENTRY.pom_order_pk)
```

```
  LEFT OUTER JOIN dbo.RVPOM_ENTRY_STATUS ON
(dbo.RVPOM_ENTRY.pom_entry_status_pk=dbo.RVPOM_ENTRY_STATUS.pom_entry_
status_pk)
WHERE
  (dbo.RVPOM_ENTRY_STATUS.id != 'Fertig')
```

Mit dieser SQL-Abfrage werden fünf Einträge ausgelesen, welche in der „SITMesDb“-Datenbank gespeichert sind und nicht den Arbeitsgangstatus “Fertig“ haben. Diese sind:

- Order ID,
- Entry ID,
- Menge (matl_qty),
- Welches Produkt (MaterialForOrder.DefName) und der
- Entry Status

Das Ereignis „DataGrid1_ItemCommand“ wird ausgelöst, wenn einer der Auswahl-Buttons im oberen „DataGrid“ gedrückt wird.

```
Private Sub DataGrid1_ItemCommand(ByVal source As Object, _
ByVal e As System.Web.UI.WebControls.DataGridCommandEventArgs) _
Handles DataGrid1.ItemCommand
    'e.Item is the row of the table where the button was clicked.
    DataGrid1.SelectedItemStyle.BackColor = Color.Yellow
    'Damit wird die ausgewählte Zeile gelb hinterlegt
    Textbox1.Text = e.Item.Cells(1).Text
    'Hier werden die fünf Text Box Elemente gefüllt mit der -
    ausgewählten Zeile des „DataGrid“s
    Textbox2.Text = e.Item.Cells(2).Text
    Textbox3.Text = e.Item.Cells(3).Text
    Textbox4.Text = e.Item.Cells(4).Text
    EntrystatusTB.Text = e.Item.Cells(5).Text

    'Hier werden die Buttons im Menü aktiviert oder deaktiviert
    If EntrystatusTB.Text = "FREIGEGEBEN" Then
        AnmeldenBtn.Enabled = True
        rüstenBtn.Enabled = False
        RüstenBeendenBtn.Enabled = False
```

```
        PaletteAGeinlegenBtn.Enabled = False
        FertigMeldungBtn.Enabled = False
    End If

    If EntrystatusTB.Text = "ANGEMELDET" Then
        AnmeldenBtn.Enabled = False
        rüstenBtn.Enabled = True
        RüstenBeendenBtn.Enabled = False
        PaletteAGeinlegenBtn.Enabled = True
        FertigMeldungBtn.Enabled = False
    End If

    If EntrystatusTB.Text = "RUESTEN" Then
        AnmeldenBtn.Enabled = False
        rüstenBtn.Enabled = False
        RüstenBeendenBtn.Enabled = True
        PaletteAGeinlegenBtn.Enabled = False
        FertigMeldungBtn.Enabled = False
    End If

    If EntrystatusTB.Text = "IN BEARBEITUNG" Then
        AnmeldenBtn.Enabled = False
        rüstenBtn.Enabled = False
        RüstenBeendenBtn.Enabled = False
        PaletteAGeinlegenBtn.Enabled = True
        FertigMeldungBtn.Enabled = False
    End If

    If EntrystatusTB.Text = "ANGEHALTEN" Then
        AnmeldenBtn.Enabled = False
        rüstenBtn.Enabled = True
        RüstenBeendenBtn.Enabled = False
        PaletteAGeinlegenBtn.Enabled = True
        FertigMeldungBtn.Enabled = True
```

```
    End If
End Sub
```

Mit Hilfe des Eigenschaftengenerators wurde eine Schaltflächenspalte im „DataGrid“ hinzugefügt. Man kann als Schaltflächentyp zwei verschiede Eigenschaften einstellen, einen „LinkButton“ oder einen „PushButton“, siehe Abbildung 9.20.

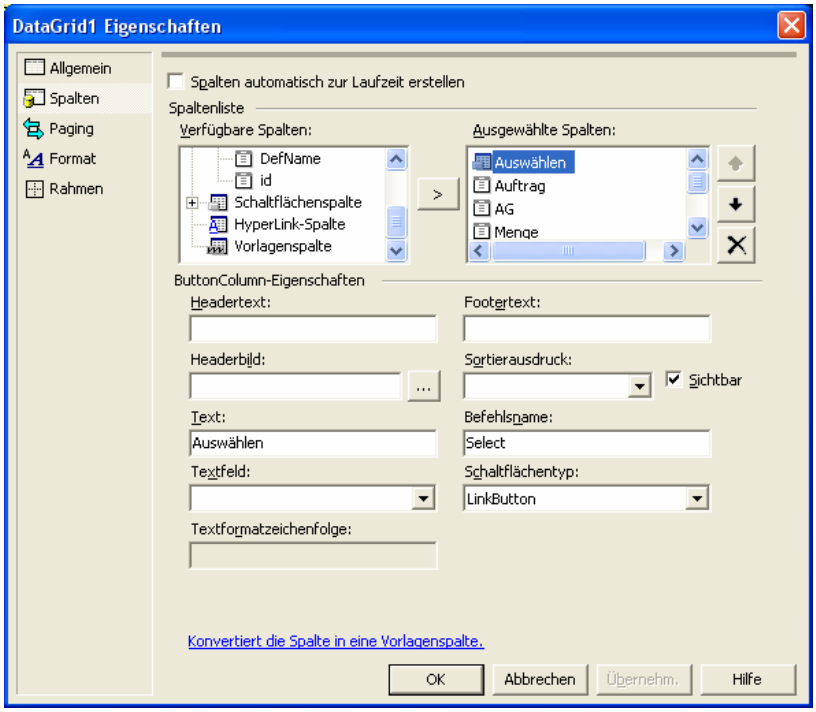

Abbildung 9.20: Einfügen einer Schaltflächenspalte

Der „PushButton“ sieht wie ein normaler Button in MS Visual Basic aus. Hier wird jedoch ein „LinkButton“ verwendet, bei dem die Auswahl durch Klicken auf den Text erfolgt, ähnlich wie bei einem Hyperlink, siehe Abbildung 9.19.

Das Ereignis „AnmeldenBtn_Click“ wird ausgelöst, wenn der Button „AnmeldenBtn“ gedrückt wird. Vorher müssen noch drei globale Variablen deklariert werden. Diese sind:

- Dim pomSrv2 As New POMSERVER2.POMApplicationSrvClas
- Dim statusresult As POMSERVER2.POM_StatusResult
- Dim x as Long

Damit wurde „pomSrv2“ als eine „POMSERVER2.POMApplicationSrvClas“-Klasse deklariert und „statusresult“ als Structure „POM_StatusResult“. Die Variable „x“ wird benötigt, da Funktionen der Klasse aufgerufen werden. Diese Funktionen liefern ein Ergebnis als Antwort welche den Variablentyp „Long“ benötigen.

Bei den folgenden Ereignissen ist der Programmcode immer sehr ähnlich. Es wird jeweils zuerst mit dem POM-Server eine Verbindungen hergestellt. Dies geschieht mit Hilfe der Funktion „Connect“, welche in der Klasse enthalten ist. Die beiden Strings “diplomand1“ und “mes2007“ werden als Übergabeparameter eingegeben und stehen für den Benutzer und das Passwort des SIMATIC IT Production Suite-Users.

Die Funktion „GetEntrySrv.SetStatus“ fordert drei Übergabeparameter. In „TextBox2.Text“ steht, wie oben erklärt, die Identität des Arbeitsgang („EntryID“), welche als erste übergeben werden muss. Als zweiter Parameter wird der gewünschte Status übergeben (als String, deshalb unter ““) und als dritter Parameter die oben global deklarierte Variable „statusresult“.

Zum Schluss wird mit Hilfe der Funktion „Disconnect“ die Verbindung zum Server wieder getrennt.

```
Private Sub AnmeldenBtn_Click(ByVal sender As System.Object, _
ByVal e As System.EventArgs) Handles AnmeldenBtn.Click
    x = pomSrv2.Connect("diplomand1", "mes2007")
    x = pomSrv2.GetEntrySrv.SetStatus(Textbox2.Text, _
    "ANGEMELDET", statusresult)
    'ErgebnisTB.Text = x
    'würde das Ergebnis der Funktion in einer TextBox anzeigen
    x = pomSrv2.Disconnect

    Response.Redirect("AG.aspx")
    'Die Seite AG.aspx wird dadurch neu geladen
End Sub

Private Sub rüstenBtn_Click(ByVal sender As System.Object, _
ByVal e As System.EventArgs) Handles rüstenBtn.Click
    x = pomSrv2.Connect("diplomand1", "mes2007")
    x = pomSrv2.GetEntrySrv.SetStatus(Textbox2.Text, _
    "RUESTEN", statusresult)
    x = pomSrv2.Disconnect

    Response.Redirect("AG.aspx")
    'Die Seite AG.aspx wird dadurch neu geladen
End Sub
```

```
Private Sub RüstenBeendenBtn_Click(ByVal sender As System.Object, _
ByVal e As System.EventArgs) Handles RüstenBeendenBtn.Click
    x = pomSrv2.Connect("diplomand1", "mes2007")
    x = pomSrv2.GetEntrySrv.SetStatus(Textbox2.Text, _
    "ANGEHALTEN", statusresult)
    x = pomSrv2.Disconnect

    Response.Redirect("AG.aspx")
    'Die Seite AG.aspx wird dadurch neu geladen
End Sub

Private Sub bearbeitenBtn_Click(ByVal sender As System.Object, _
ByVal e As System.EventArgs)
    x = pomSrv2.Connect("diplomand1", "mes2007")
    x = pomSrv2.GetEntrySrv.SetStatus(Textbox2.Text, _
    "IN BEARBEITUNG", statusresult)
    x = pomSrv2.Disconnect

    Response.Redirect("AG.aspx")
    'Die Seite AG.aspx wird dadurch neu geladen
End Sub

Private Sub FertigMeldungBtn_Click(ByVal sender As System.Object, _
ByVal e As System.EventArgs) Handles FertigMeldungBtn.Click
    x = pomSrv2.Connect("diplomand1", "mes2007")
    x = pomSrv2.GetEntrySrv.SetStatus(Textbox2.Text, _
    "FERTIG", statusresult)
    x = pomSrv2.Disconnect

    Response.Redirect("AG.aspx")
    'Die Seite AG.aspx wird dadurch neu geladen
End Sub
```

```
Private Sub Zurück3_Click(ByVal sender As System.Object, _
ByVal e As System.EventArgs) Handles Zurück3.Click
    Response.Redirect("DemoCab.aspx")
    'Die Hauptmenü- Seite DemoCab.aspx wird geladen
End Sub
```

Ein Button wurde bis jetzt noch nicht beschrieben. Er soll das Einlegen der Palette in die Be-/Entladestation bestätigen und definieren, zu welchem AG die Palette gehört. Man benötigt für diesen Button keine Zeilen im Programmcode, da es sich um einen CAB-Button handelt. Dieser hat zusätzliche Eigenschaften, welche mit einem eigenen Icon im Fenster „Eigenschaften“ sichtbar werden, siehe Abbildung 9.21.

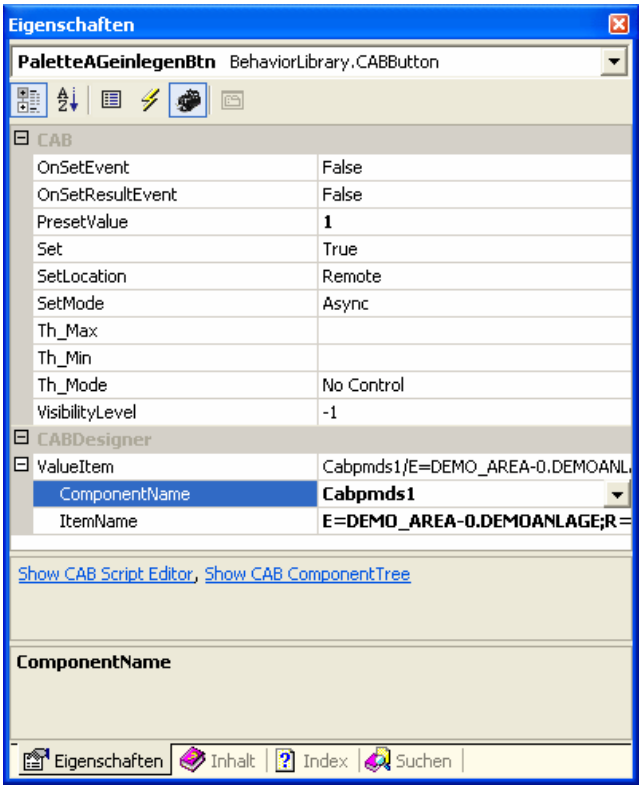

Abbildung 9.21: CAB-Eigenschaften

Um auf Daten des Production Modelers zugreifen zu können muss man ein „Cabpmds“ Element aus der CAB Library einfügen, siehe Abbildung 9.22. Die Abkürzung „Cabpmds“ steht für „Client Application Builder Production Modeler Data Source“. Dieses Element stellt die Verbindung zwischen MS Visual Basic und dem PM her.

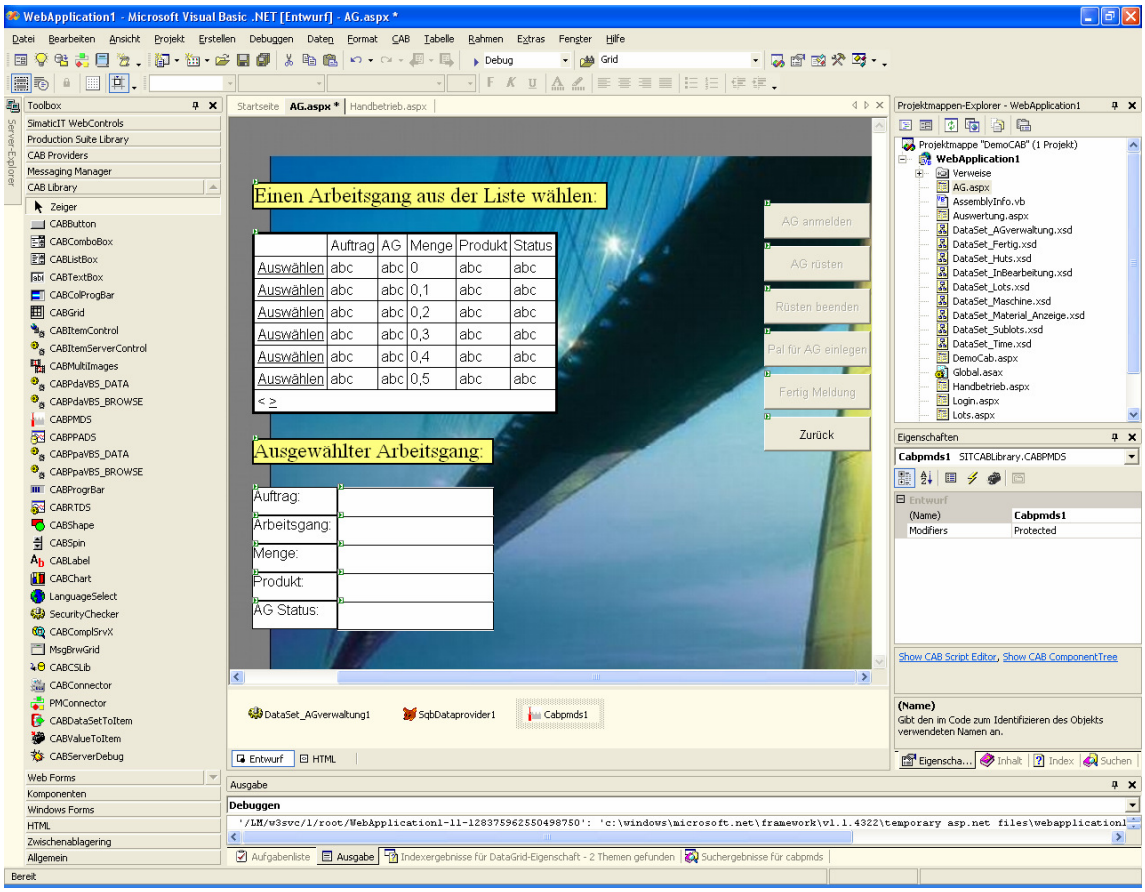

Abbildung 9.22: „Cabpmds“-Element hinzufügen

Der CAB-Button braucht noch folgende Einstellungen in den CAB-Eigenschaften damit er eine Regel im PM startet, siehe auch Abbildung 9.21:

- „PresetValue“ auf „1“ setzen, dadurch startet der Button die gewünschte Regel (Standardeinstellung ist jedoch „0“)
- Bei der Eigenschaft „Component Name“ muss man das Cabpmds Objekt auswählen.
- Danach kann man bei der Eigenschaft „Item Name“ folgendes Fenster zur Auswahl öffnen.

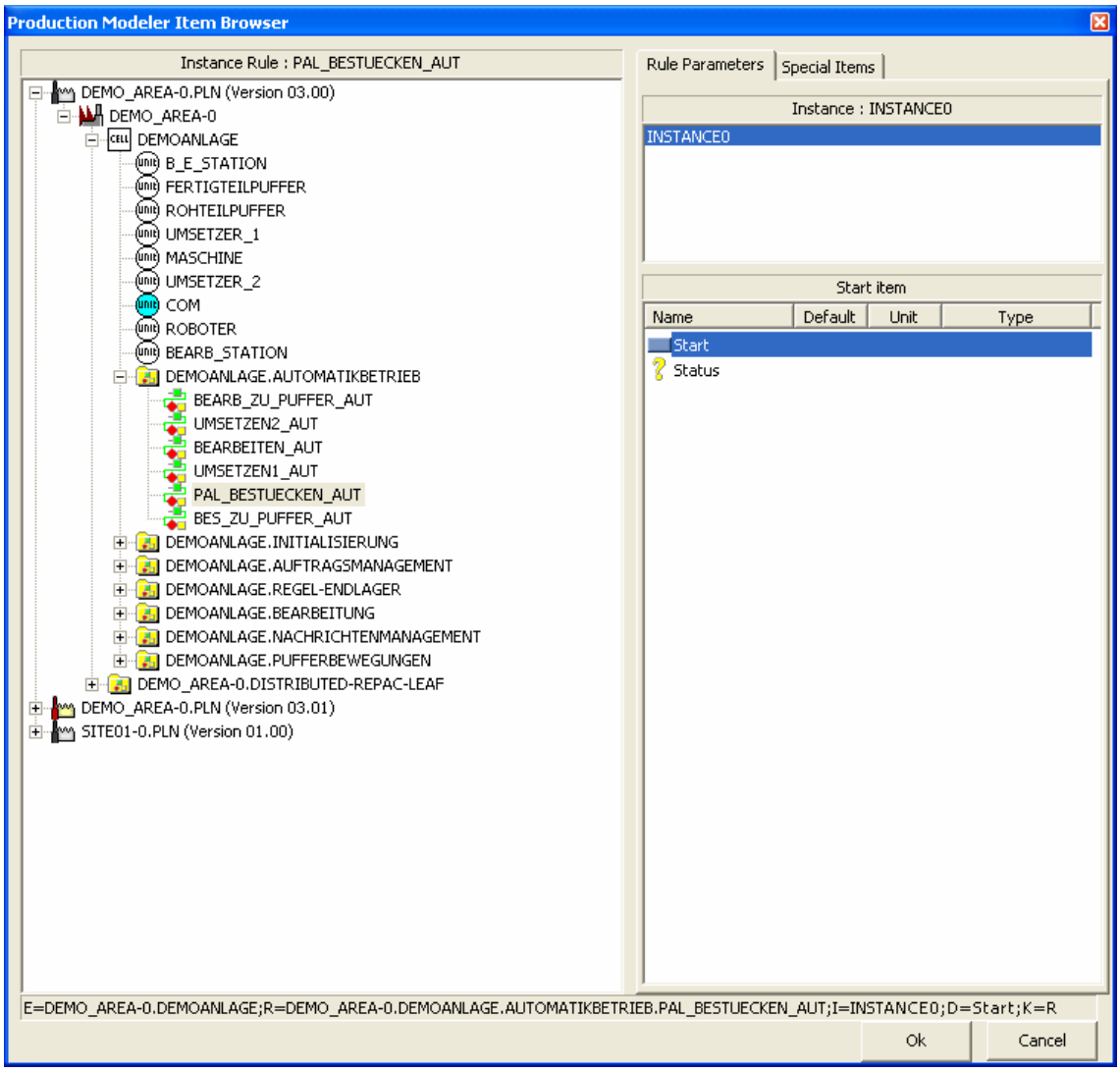

Abbildung 9.23: Regelzuweisung beim CAB-Button

Im Fenster „Production Modeler Item Browser“ kann man nun die vorhandene Anlage nach Regeln durchsuchen. Die Regel, die unser CAB-Button starten soll, befindet sich in der Cell „DEMOANLAGE“ im Ordner „DEMOANLAGE.AUTOMATIK-BETRIEB“, welcher im Production Modeler erstellt wurde. Die auszuwählende Regel heißt „PAL_BESTUECKEN_AUT“. Nach dem Auswählen muss im rechten Bereich im Teilfenster „Start Item“ der „Start“-Button markiert werden, siehe Abbildung 9.23. Im Bereich „Instance“ muss durch Klicken der rechten Maustaste – Add eine Instance hinzugefügt werden. Mit dem „OK“-Button werden die Eingaben bestätigt und gespeichert.

Zusammenfassend kann man sagen, dass auf der Seite „Arbeitsgang Verwaltung“ alle, für die Handhabung der Anlage, wichtigen Elemente enthalten sind. Falls Probleme in der Anlage auftreten und händisch in den Ablauf eingegriffen werden muss, ist ein Handbetrieb vorgesehen, welcher die Einzelschritte (Verfahrbewegungen) starten kann.

9.4.4 Das Web-Formular „Handbetrieb“

Das Web-Formular „Handbetrieb“ ist zur manuellen Steuerung der Verfahrbewegungen der Anlage gedacht. Beim Auftreten eines Fehlers in der Anlage und einem damit verbundenen Abbruch des Automatikbetriebes, soll der Anwender die Möglichkeit haben, die Verfahrbewegungen manuell zu starten. Eine Art Einzelschrittmodus für den Transport der Paletten von einer Station zur Anderen. Hierfür wurde das Web-Formular „Handbetrieb.aspx“ entwickelt, siehe Abbildung 9.24.

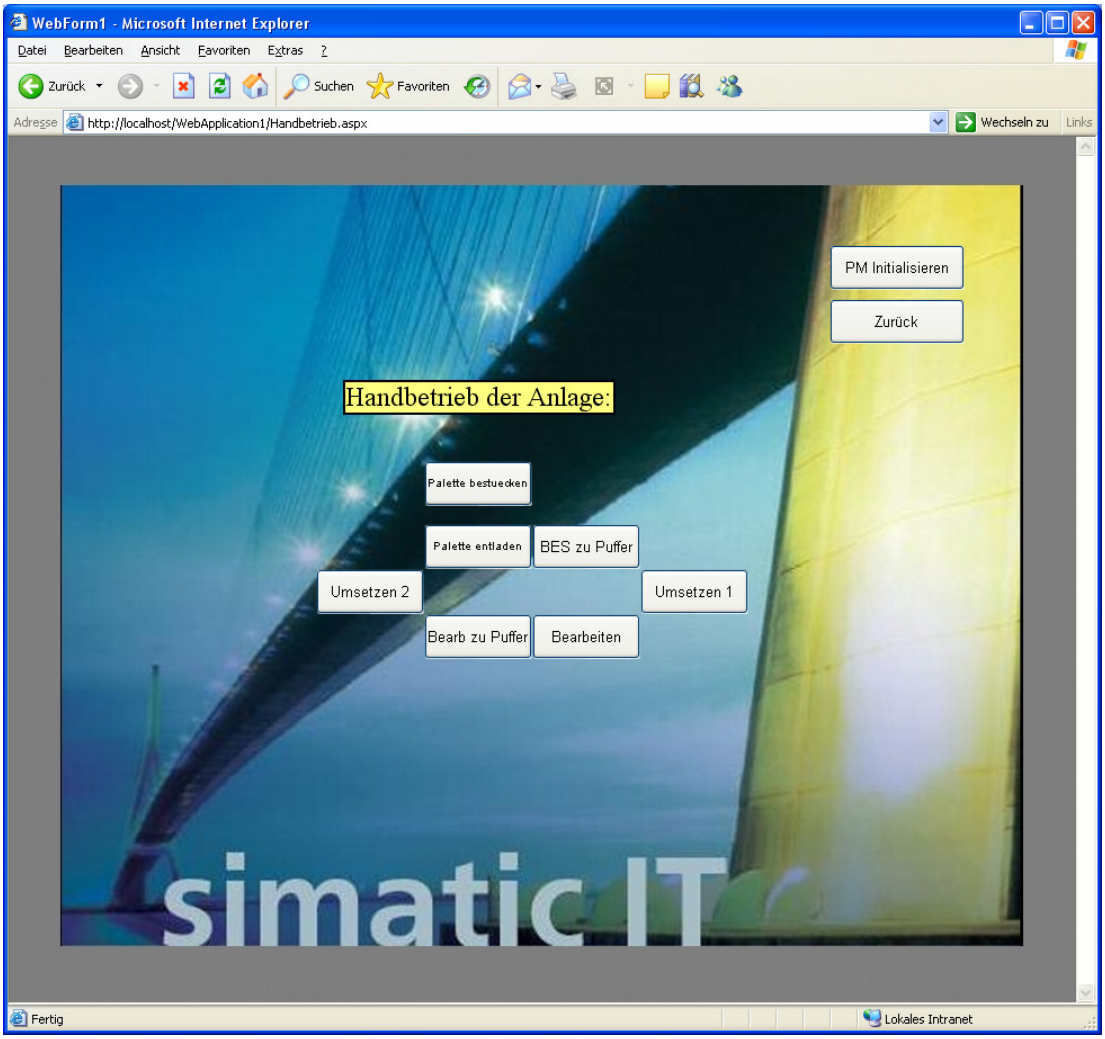

Abbildung 9.24: Web-Formular „Handbetrieb“

Auf dieser Seite befinden sich mehrere CAB-Buttons, die jeder für sich eine Regel im Production Modeler aufrufen. In der folgenden Tabelle sind die Buttons und die dazugehörigen Regeln, die im Production Modeler aufgerufen werden, aufgelistet.

CAB Button	im PM aufgerufene Regel
PM Initialisieren	INITIALISIERUNG
Palette bestuecken	PALETTE_BESTUECKEN
Palette entladen	PALETTE_ENTLADEN
BES zu Puffer	BES_ZU_PUFFER
Umsetzen 1	UMSETZEN_1
Bearbeiten	FERTIGEN
Bearb zu Puffer	BEARBSTATION_ZU_PUFFER
Umsetzen 2	UMSETZEN_2

Abbildung 9.25: Zuordnung der CAB-Buttons zu den PM-Regeln

Die richtige Konfiguration der CAB-Eigenschaften wird in Kapitel 9.4.3 beschrieben. Das Menü dieser Seite (rechts) enthält nur zwei Buttons, einen um wieder auf die Seite „Hauptmenü“ zu gelangen und einen CAB-Button „PM Initialisieren“. Dieser startet im PM eine Regel, welche die Zustände der Anlage auf den Ausgangszustand zurücksetzt, d.h. es befindet sich keine Palette in der Anlage. Der Programmcode beschränkt sich auf den „Zurück“ Button und sieht wie folgt aus:

```
Private Sub Zurück3_Click(ByVal sender As System.Object, -
ByVal e As System.EventArgs) Handles Zurück3.Click
    Response.Redirect("DemoCab.aspx")
End Sub
```

Das Ereignis „Zurück3_Click“ wird ausgelöst, wenn man auf den Button klickt und die Befehlszeile „Response.Redirect("DemoCab.aspx")“ bewirkt das Laden der Seite „DemoCab.aspx“, welche das Hauptmenü darstellt.

9.4.5 Das Web-Formular „Auswertung“

Das Web-Formular „Auswertung“ dient zur zeitlichen Auswertung eines Arbeitsgangs der bereits fertig gemeldet ist. Die Seite sieht wie folgt aus:

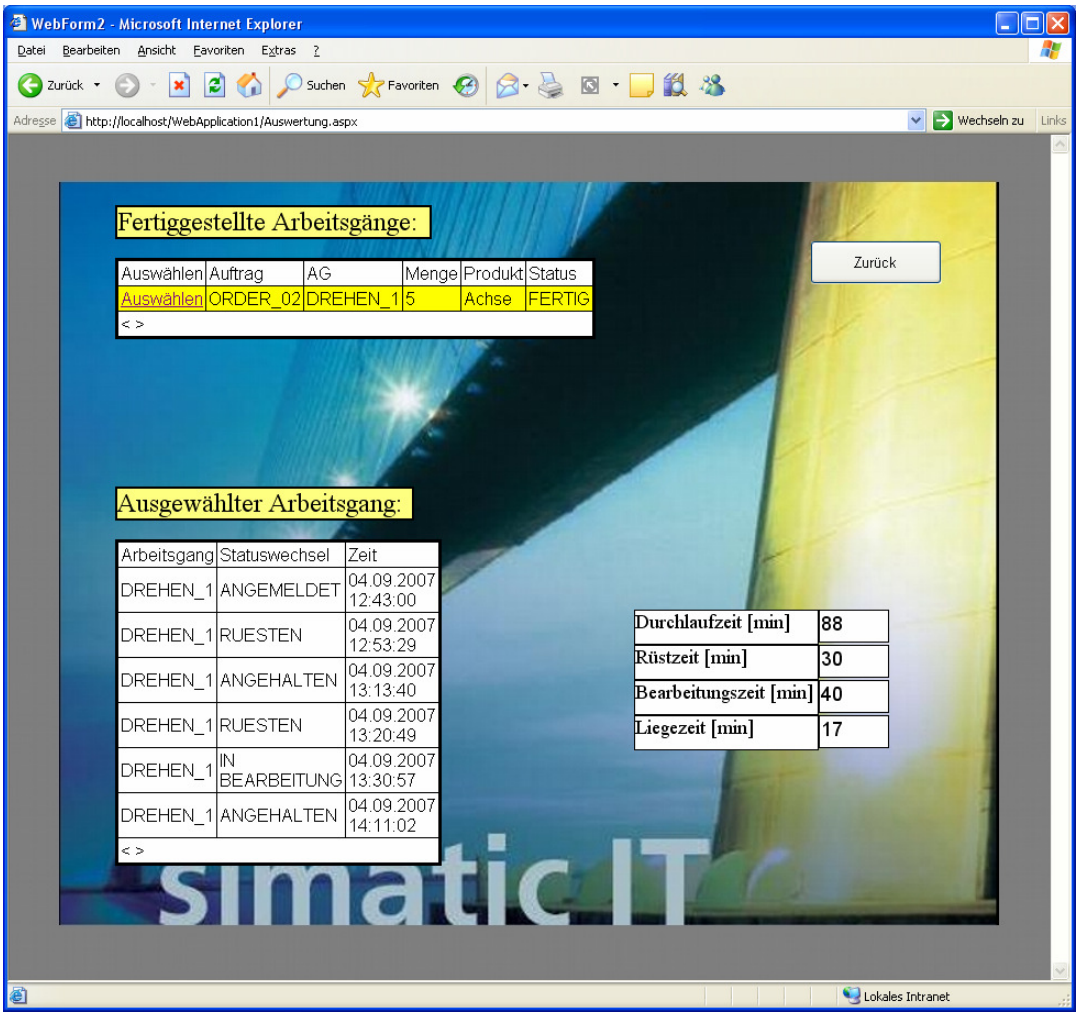

Abbildung 9.26: Web-Formular „Auswertung“

Im Menü befindet nur ein Button um in das Hauptmenü zurückzukehren. Im linken oberen Bereich der Seite wird ein „DataGrid“ angezeigt, welches mit Hilfe einer geeigneten SQL-Abfrage alle fertigen Arbeitsgänge anzeigt. Weiters enthält das „DataGrid“ eine Schaltflächenspalte zur Auswahl eines Eintrages. Wählt man einen Eintrag durch Drücken des „LinkButtons“ im „DataGrid“ aus, wird im zweiten „DataGrid“, links unterhalb, eine Liste mit allen Statuswechseln des Arbeitsgangs eingeblendet. Rechts daneben befinden sich dann Text-Box-Elemente zur Anzeige der ausgewerteten Zeiten. Die Daten hierfür werden vom SIMATIC IT Order Manager, bei jedem Statuswechsel, in die SQL Datenbank abgelegt. Die Berechnung der Zeiten erfolgt dann in der Prozedur „Private Sub DataGrid2_ItemCommand“. Der Programmcode für das Web Formular sieht wie folgt aus:

Diese Variablen werden global in der Klasse deklariert und in den Unterprogrammen verwendet.

```
Dim entry As String
```

```
Dim rüsten, bearbeiten, liege As Long
Dim i, zähler As Integer
Dim zeit, zeit2 As Date
```

Das Ereignis „Page_Load“ wird gestartet, wenn die Seite neu geladen wird. Die Zeile „SqbDataprovider2.Fill(DataSet_Fertig1)“ füllt den Datensatz „DataSet_Fertig1“ mit den neuen Werten aus der SQL-Datenbank und die Zeile „DataGrid2.DataBind()“ speichert (bindet) diese Werte im DataGrid2, siehe auch Kapitel 9.4.2.

```
Private Sub Page_Load(ByVal sender As System.Object, -
ByVal e As System.EventArgs) Handles MyBase.Load
    SqbDataprovider2.Fill(DataSet_Fertig1)
    DataGrid2.DataBind()
End Sub
```

Die SQL-Abfrage für den Datensatz „DataSet_Fertig1“ sieht folgendermaßen aus:

```
SELECT
  dbo.RVPOM_ORDER.pom_order_id,
  dbo.RVPOM_ENTRY.pom_entry_id,
  dbo.RVPOM_ORDER.pom_matl_qty,
  MaterialForOrder.DefName,
  dbo.RVPOM_ENTRY_STATUS.id
FROM
  dbo.RVMM_MATERIAL_INFO  MaterialForOrder RIGHT OUTER JOIN
dbo.RVPOM_ORDER ON
(MaterialForOrder.DefID=dbo.RVPOM_ORDER.matl_def_id)
   LEFT OUTER JOIN dbo.RVPOM_ENTRY ON
(dbo.RVPOM_ORDER.pom_order_pk=dbo.RVPOM_ENTRY.pom_order_pk)
   LEFT OUTER JOIN dbo.RVPOM_ENTRY_STATUS ON
(dbo.RVPOM_ENTRY.pom_entry_status_pk=dbo.RVPOM_ENTRY_STATUS.pom_entry_
status_pk)
WHERE
   (dbo.RVPOM_ENTRY_STATUS.id = 'FERTIG')
```

Mit dieser SQL-Abfrage werden fünf Einträge ausgelesen, welche in der „SITMesDb“ Datenbank gespeichert sind und den Arbeitsgangstatus “Fertig“ haben. Diese sind:

- Order ID,
- Entry ID,
- Menge (matl_qty),
- Welches Produkt (MaterialForOrder.DefName) und der
- Entry Status

Nur diese Arbeitsgänge sind für die Auswertung der Zeiten interessant, da keine Statuswechsel mehr hinzukommen.

Das Ereignis „DataGrid2_ItemCommand“ wird ausgelöst, wenn einer der Auswahl-Buttons im „DataGrid“ gedrückt wird.

```
Private Sub DataGrid2_ItemCommand(ByVal source As Object, ByVal e As
System.Web.UI.WebControls.DataGridCommandEventArgs) Handles
DataGrid2.ItemCommand
    DataGrid2.SelectedItemStyle.BackColor = Color.Yellow
    'Damit wird die ausgewählte Zeile gelb hinterlegt
    entry = e.Item.Cells(2).Text
    'Hier wird der Entry Eintrag aus der SQL auf die Variable entry
    'geschrieben
    SqbDataprovider1.Query = "SELECT entry_id AS Expr1, -
    status_id AS Expr2, ts AS Expr3 FROM POM_STATUS_LOG -
    WHERE (entry_id = '" & entry & "')"
    'Hier wird die Query Eigenschaft des SqbDataprovider1 mit einer
    'SQL-Abfrage belegt, wo entry als Variable hinzugefügt wird
    SqbDataprovider1.Fill(DataSet_Time1)
    'DataSet_Time1 wird gefüllt aus der SQL-Datenbank mit der oben
    'festgelegten Abfrage
    DataGrid1.DataBind()
    'In das DataGrid1 werden diese Daten gespeichert

    'Hier erfolgt die Berechnung der Zeiten:
    rüsten = 0
    bearbeiten = 0
    liege = 0
```

```
        zähler = DataGrid1.Items.Count
        'Anzahl der ausgelesenen daten im DataGrid1
        i = 0

        zeit = DataGrid1.Items.Item(0).Cells(2).Text
        zeit2 = DataGrid1.Items.Item(zähler - 1).Cells(2).Text
        dlzTB.Text = DateDiff(DateInterval.Minute, zeit, zeit2)
        'Hier wird die Differenz zwischen den beiden Einträgen
        'ermittelt und an die Text Box dlzTB geschickt

    'In diesem Programmabschnitt werden die jeweiligen Einträge den
    'richtigen Zeiten zugeordnet
    Do While zähler > i + 1
        zeit = DataGrid1.Items.Item(i).Cells(2).Text
        zeit2 = DataGrid1.Items.Item(i + 1).Cells(2).Text
    If DataGrid1.Items.Item(i).Cells(1).Text = "ANGEMELDET" Then
        liege = liege + DateDiff(DateInterval.Minute, zeit, zeit2)
    Else
    If DataGrid1.Items.Item(i).Cells(1).Text = "RUESTEN" Then
        rüsten = rüsten + DateDiff(DateInterval.Minute, zeit, zeit2)
    Else
    If DataGrid1.Items.Item(i).Cells(1).Text = "IN BEARBEITUNG" Then
        bearbeiten = bearbeiten + DateDiff(DateInterval.Minute, -
        zeit, zeit2)
    Else
    If DataGrid1.Items.Item(i).Cells(1).Text = "ANGEHALTEN" Then
        liege = liege + DateDiff(DateInterval.Minute, zeit, zeit2)
    End If
    End If
    End If
    End If
    'Hier werden alle Verzweigungen geschlossen.
    i = i + 1
    Loop
```

```
    'Ab dieser Stelle wird die While Schleife wiederholt durchlaufen
    'bis die Ausstiegsbedinngung erfüllt ist.

    'Hier werden die jeweils ermittelten Zeiten auf die dafür
    'vorgesehene Text Box geschrieben.
    bearbeitenTB.Text = bearbeiten
    rüstenTB.Text = rüsten
    liegeTB.Text = liege
    End Sub
```

Das Ereignis „Zurück4_Click“ wird ausgelöst, wenn man auf den Button „Zurück“ klickt und durch die Befehlszeile „Response.Redirect("DemoCab.aspx")“ wird das Laden der Seite „DemoCab.aspx“ ausgelöst.

```
    Private Sub Zurück4_Click(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles Zurück4.Click
        Response.Redirect("DemoCab.aspx")
    End Sub
```

Das Web-Formular „Auswertung.aspx“ ist also ein Informationswerkzeug und dient nicht zur Steuerung der Anlage. Mit Hilfe dieser SQL-Abfrage können alle gewünschten Daten, die in der SQL-Datenbank hinterlegt sind, ausgelesen und angezeigt werden.

9.4.6 Das Web-Formular „Lots/Sublots“

Dieses Web-Formular dient ebenfalls zur Anzeige von Werten, welche in der SQL-Datenbank hinterlegt sind und sieht wie folgt aus:

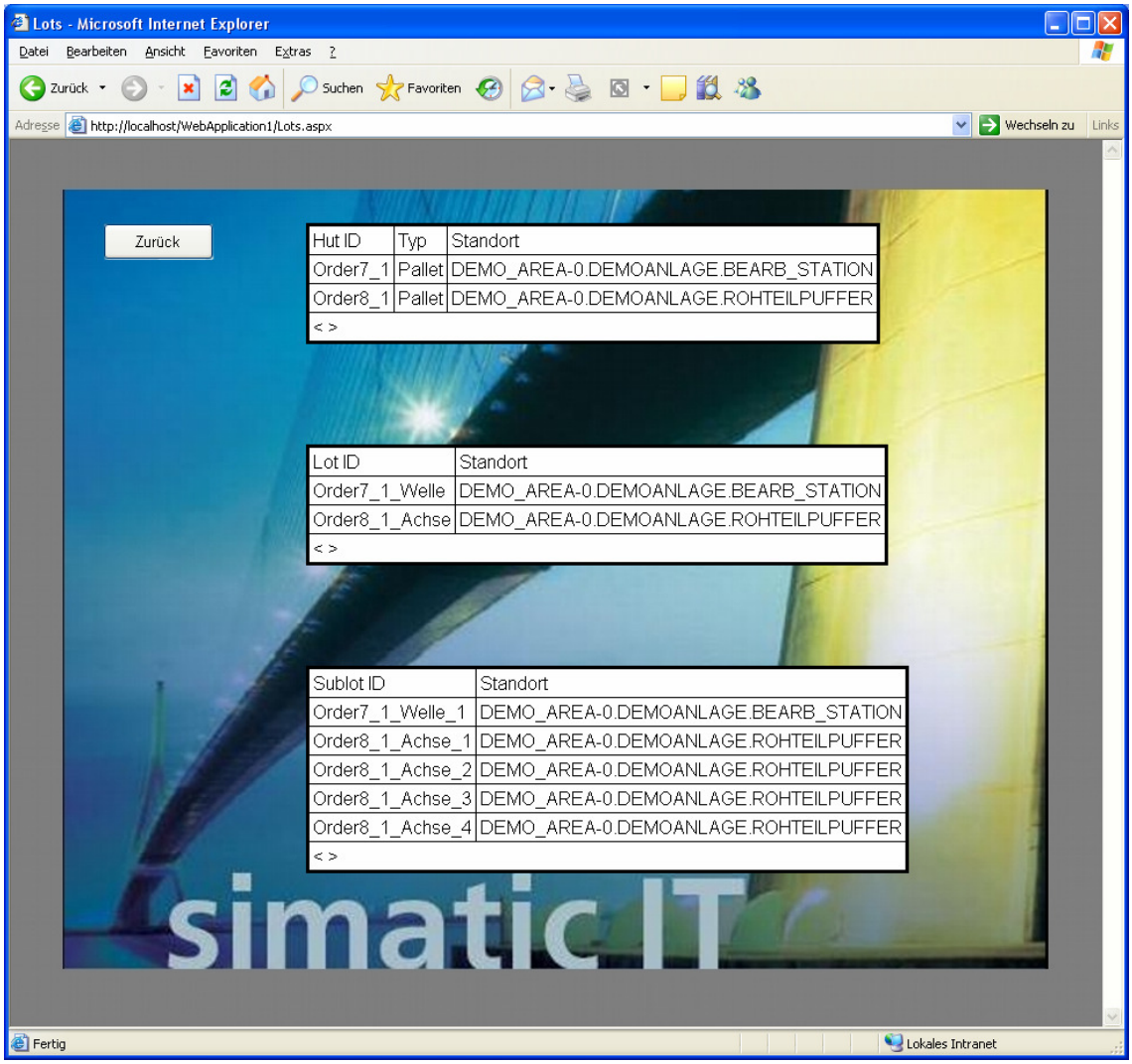

Abbildung 9.27: „Lots/Sublots-DataGrid“

Diese Fenster besteht aus jeweils drei „DataGrids“, welche mit drei Datensätzen gefüllt werden. Die Datensätze sind mit „SqbDataprovider“ generiert worden. Sie holen sich die Einträge aus der SQL-Datenbank, siehe auch Kapitel 9.4.2. Im ersten „DataGrid“ werden die Handling-Units, im Zweiten die Lots und im Dritten die Sublots angezeigt und der jeweilige Standort dieser. Der SIMATIC IT Material Manager (MM) verwaltet die Materialien und stellt Methoden für diese zur Verfügung. Im Production Modeler werden die Methoden des MM verwendet und in Regeln aufgerufen, so wird die Materialverwaltung auch über den PM mitgesteuert. Hier, in dieser Benutzeroberfläche, kann man mit Hilfe der SQL-Datenbank den IST-Zustand der Anlage abfragen. Für manuelle Eingriffe kann man die Benutzeroberfläche des Material Managers verwenden.

10 COM-Wrapper

10.1 Aufgabenstellung

Der COM-Wrapper soll Methoden und Events zur Verfügung stellen, die vom Production Modeler aus aufgerufen werden können. Sie dienen zur Abfrage und Veränderung von Attributen, also von Variablen der Anlage. Sie werden als Schnittstelle zwischen MES und der Steuerung der Anlage (SPS, NC) verwendet. Für den Simulationsbetrieb werden die Werte also auf einer SQL-Datenbank hinterlegt und von dort auch für den Simulator entnommen, siehe Kapitel 11. Die Umsetzung soll mit Hilfe einer DLL-Datei durchgeführt werden. DLL steht für „Dynamik Link Library“, was nichts anderes heißt als „Dynamische Verbindungsbibliothek“. Prozeduren, die in einer DLL programmiert werden, können wie eine Drittanbieterkomponente im Production Modeler integriert werden. Die Verwendung im Production Modeler ist in Punkt 7.3.2 beschrieben.

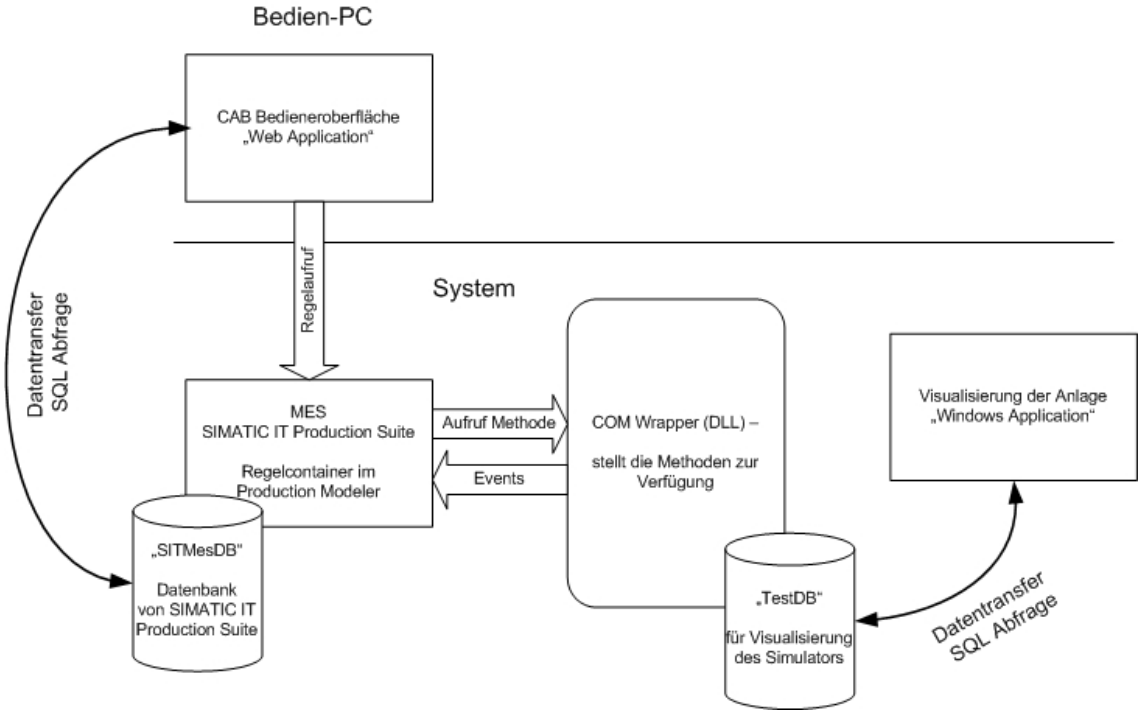

Abbildung 10.1: Systemaufbau

10.2 Beschreibung des Programmcodes der DLL

In diesem Kapitel wird der Programmcode, d.h. einige Methoden der DLL beschrieben. Den vollständigen Code aller Methoden findet man im Anhang A: Programmcode.

Zu Beginn des Programms werden globale Variablen deklariert, welche zum Auslesen eines Recordsets, d.h. eines Eintrages aus der SQL-Datenbank, benötigt werden.

```
Dim ssql As String
Dim rs As ADODB.Recordset
Dim strStream As ADODB.Stream
Dim x As Variant
Dim i As Double
Dim ConnUnt As ADODB.Connection
```

Anschließend werden globale Variablen, welche zur Übergabe der Werte aus der SQL-Datenbank dienen, deklariert. Es ist auf die korrekten Datentypen zu achten.

```
Dim gbes_zu As Boolean, gbes_a As Boolean, gbes_bel As Boolean, – usw.
```

Stellvertretend für alle Methoden werden nun Zwei genauer beschrieben.

10.2.1 Methode zur Bewegung einer Palette

Die erste Methode dient zur Bewegung einer Palette von der Be-/Entladestation zum Rohteilpuffer. Sie ist stellvertretend für alle Bewegungsmethoden und beinhaltet alle wesentlichen Schritte für die Ausführung.

```
Public Sub beszupuffer()

    Set ConnUnt = New ADODB.Connection
    ConnUnt.Provider = "sqloledb.1"
    ConnUnt.Properties("Data Source").Value = "eftpeh"
    ' Der SQL Server läuft im Rechner mit den Namen eftpeh
    ConnUnt.Properties("Initial Catalog").Value = "TestDB"
    ' Die Datenbank heißt TestDB
    ConnUnt.Properties("Current Language").Value = "German"
    'Die Sprache ist auf Deutsch eingestellt
```

```
ConnUnt.Properties("User ID") = "sa"
' SQL Username ist sa
ConnUnt.Properties("Password") = "mes2007"
' SQL Passwort ist mes2007
ConnUnt.CursorLocation = adUseClient

ConnUnt.Open

ssql = "SELECT * FROM MES2 "
Set rs = New ADODB.Recordset
rs.Open ssql, ConnUnt, adOpenKeyset, adLockOptimistic

' Hier werden nun alle Atribute so gesetzt damit die Bewegung
' richtig ablaufen kann
gbes_zu = False
rs.Fields("gbes_zu") = gbes_zu
groh_zu = False
rs.Fields("groh_zu") = groh_zu
groh_a = True
rs.Fields("groh_a") = groh_a
gbes_a = True
rs.Fields("gbes_a") = gbes_a

' Mit dem Befehl rs.Update werden dann die Attribute wieder in die
' SQL-Datenbank geschrieben und der Simulator wird aktialisiert.
rs.Update
'um den AKTIV Zusatand in der DB zu aktualisieren

' Da die Anlage eine Palette nicht wirklich von einem Punkt zum
' anderen transportiert ist auch kein Zeitraum hierfür vorhanden,
' deshalb haben wir mit Hilfe einer Message Box ein Delay erzeugt
MsgBox "BES zu Rohteilpuffer."    'statt Delay
```

```
    ' Danach werden wieder die Attribute so gesetzt als wäre die
    ' Bewegung bereits abgeschlossen

    ' Über eine Rotation wird das Attribut Paletten ID von der
    ' Be-/Entladestation auf den Rohteilpuffer übergeben
    gbes_palid = rs.Fields("gbes_palid")
    groh_palid = gbes_palid
    gbes_palid = ""
    rs.Fields("gbes_palid") = gbes_palid
    rs.Fields("groh_palid") = groh_palid
    ' Code nicht vollständig!
    ' Ähnliches passiert auch mit allen den anderen Attributen

    ' Hier wird die SQL-Datenbank wieder aktualisiert.
    rs.Update
    rs.Close
    Set rs = Nothing
    ConnUnt.Close
    Set ConnUnt = Nothing
End Sub
```

10.2.2 Methode um den aktuellen Zustand der Anlage abzufragen

Diese Methode ist, im Vergleich zur Ersten, mit Übergabeparameter versehen und dient dazu, die aktuellen Zustände der Anlage abzufragen und rückzumelden. Diese Übergabeparameter werden zwischen den runden Klammern im Methodennamen deklariert, siehe unten.

```
Public Sub Zustandpr(bes_zu As Boolean, bes_a As Boolean, usw.)

    Set ConnUnt = New ADODB.Connection
    ConnUnt.Provider = "sqloledb.1"
    ConnUnt.Properties("Data Source").Value = "eftpeh"
    ConnUnt.Properties("Initial Catalog").Value = "TestDB"
    ' Datenbank
    ConnUnt.Properties("Current Language").Value = "German"
```

```
    'con_op.SQL_Language  ' German

    ConnUnt.Properties("User ID") = "sa"
    ' SQL Username
    ConnUnt.Properties("Password") = "mes2007"
    ' SQL Passwort
    ConnUnt.CursorLocation = adUseClient

    ConnUnt.Open

    ssql = "SELECT * FROM MES2 "
    Set rs = New ADODB.Recordset
    rs.Open ssql, ConnUnt, adOpenKeyset, adLockOptimistic

    ' Anschließend werden dann alle Felder auf die Übergabeparameter
    ' geschrieben, welche im Methodenkopf definiert wurden
    bes_zu = rs.Fields("gbes_zu")
    bes_a = rs.Fields("gbes_a")
    bes_bel = rs.Fields("gbes_bel")
    bes_palid = rs.Fields("gbes_palid") usw.

    ' Die Verbindung zum SQL Server wird wieder geschlossen und das
    ' Recordset wird auf Nothing gesetzt
    rs.Close
    Set rs = Nothing
    ConnUnt.Close
    Set ConnUnt = Nothing

End Sub
```

Die Daten können nach Aufruf dieser Methode im Production Modeler verwendet werden und dienen auch als Sicherheitsabfragen in den Regeln. Mit dieser Art von Parameterübergabe kann man auch Daten zu einer realen Anlage senden und empfangen. Das Abfragen von Merkern einer SPS sowie Signalausgängen einer DI/DO-Karte kann ebenfalls so realisiert werden.

11 Simulator mit Anlagenvisualisierung

11.1 Aufgabenstellung

Die reale Anlage soll mit Hilfe eines Simulators abgebildet werden um die Auswirkungen auf die Anlage durch das MES zu beobachten. Da keine realen Signale von der Anlage abgefragt werden können, soll eine SQL-Datenbank verwendet werden. Mit deren Hilfe können Zustände abgerufen, verändert und auch wieder gespeichert werden. Der Simulator soll also den aktuellen Zustand der Anlage wiedergeben und kann auch von einem anderen PC im Netzwerk gestartet und ausgeführt werden.

11.2 Allgemeines

In diesem Kapitel wird beschrieben, wie der Simulator, also das Programm für die Visualisierung der Anlage, programmiert wird. Die Oberfläche zeigt im Wesentlichen die Förderbänder der Anlage und den Inhalt der Paletten, die sich auf den Arbeitsplätzen befinden. Ob die Hubzylinder der Förderstrecke aus- oder eingefahren sind, wird mit einem Textfeld angezeigt. Eine 3D Programmierung wäre für diesen Zweck einen zu großen Aufwand. Man sieht die Förderstrecke also von der Vogelperspektive, siehe Abbildung 11.1. Der Roboter und die Maschine werden nur eingeblendet wenn sie gerade im Einsatz (aktiv) sind.

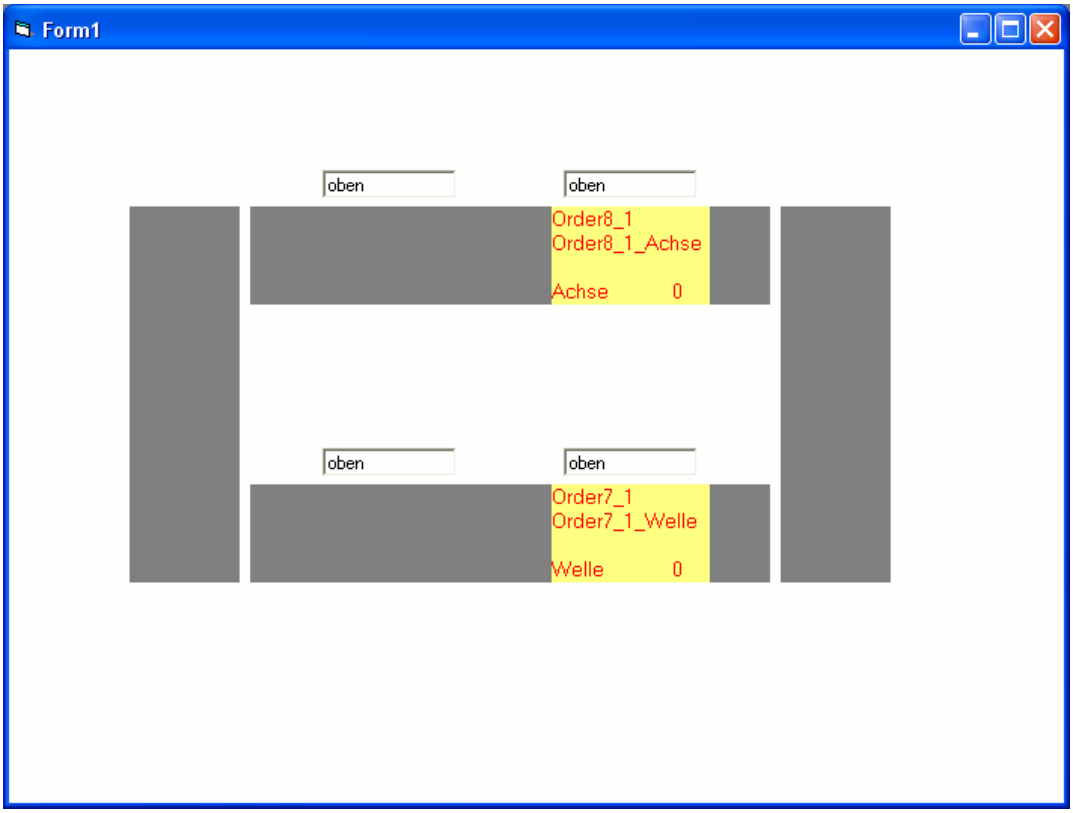

Abbildung 11.1: Visualisierung der Anlage

11.3 Beschreibung des Simulators

Der Simulator ist ein Programm, mit dem Werte aus einer SQL-Datenbank in kurzen und regelmäßigen Intervallen ausgelesen werden. Diese können auf verschiedene Art und Weise angezeigt werden. Entweder durch farbliche Veränderungen, durch Textfelder oder durch Symbole, welche ein- und ausgeblendet werden. Als Software wurde MS Visual Basic 6.0 verwendet. Die Erläuterungen sind direkt im Code hinterlegt um für spätere User das Programm verständlicher zu gestalten. Die Oberfläche des Simulators (Form) beinhaltet folgende Elemente:

- Timer
- Text Box
- Label
- Picture Box

Alle Elemente wurden mit „Drag and Drop“ in das Form eingefügt und so angeordnet, wie in Abbildung 11.2 zu sehen ist.

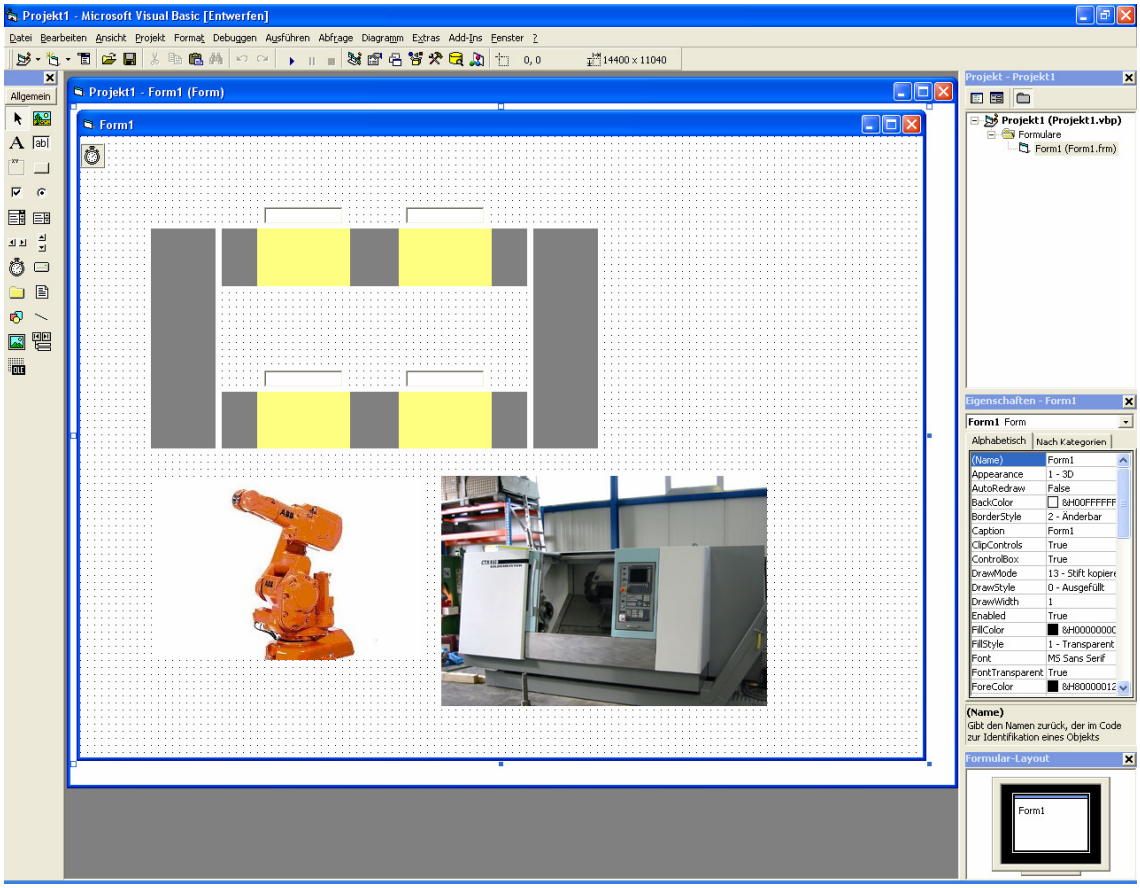

Abbildung 11.2: Oberfläche des Simulators

Die Picture-Box-Elemente werden verwendet um den Roboter sowie die Maschine darzustellen. Man kann diesem Element ein Bild hinterlegen und über die Eigenschaft „Visible“ kann man das Element ein- und ausblenden. Somit erhält man die Möglichkeit den Roboter oder auch die Maschine einzublenden wenn sie gerade aktiv ist. Die Handhabung erfolgt über den Programmcode, der etwas später im Kapitel beschrieben wird.

Der Timer ist für die regelmäßige Abfrage der Datenbank, sowie die Aktualisierung der Anzeige verantwortlich. Über die Eigenschaft „Intervall“ kann die Frequenz des Timers eingestellt werden, mit der er bestimmte Dinge ausführt. Die Angabe erfolgt in [ms] und ist hier auf „100“ eingestellt, d.h. alle 0,1 Sekunden wird die Datenbank abgefragt. Das Unterprogramm, das in diesem Intervall gestartet wird, heißt „Private Sub Timer1_Timer()“ und wird im Programmcode beschrieben.

Um einige Befehle zum Auslesen eines Recordsets verwenden zu können, müssen Verweise auf DLL-Dateien gemacht werden. Damit kann man die Unterprogramme dieser Objekte verwenden und auch manche Variablendeklarationen durchführen. In Abbildung 11.3 sieht man die Liste der Verweise, welche man für dieses Programm benötigt.

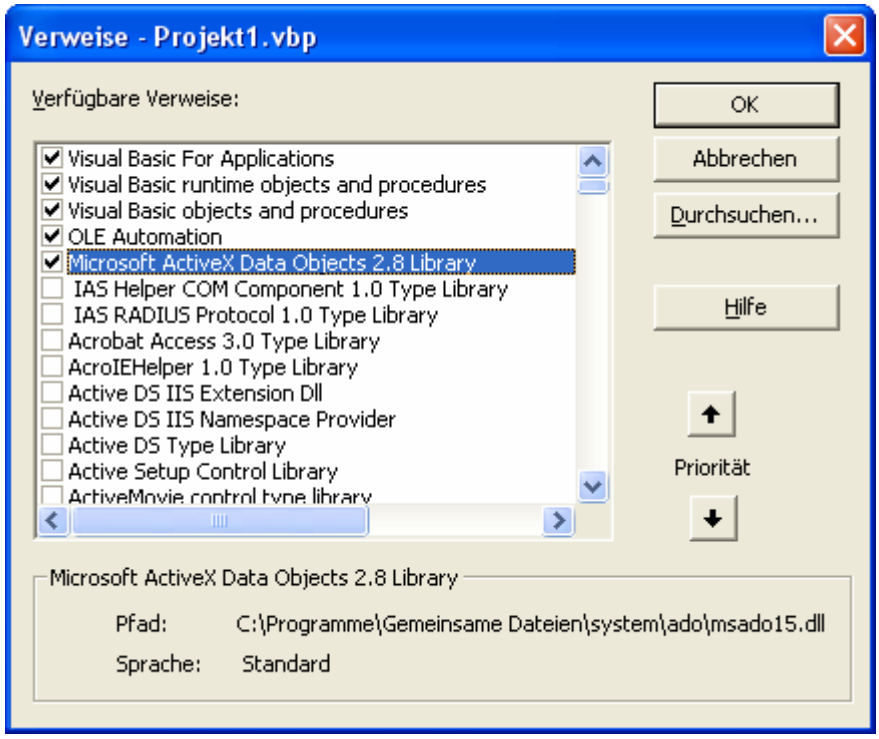

Abbildung 11.3: Verweise auf DLL-Dateien für das Simulatorprogramm

Folgende globalen Variablen werden zu Beginn des Programms deklariert. Sie werden zum Auslesen eines Recordsets, d.h. eines Eintrages aus der SQL-Datenbank benötigt, siehe später im Programmcode.

```
Dim ssql As String
Dim rs As ADODB.Recordset
Dim strStream As ADODB.Stream
Dim x As Variant
Dim i As Double
Dim ConnUnt As ADODB.Connection
```

Die folgenden Variablen dienen als Zwischenspeicher im Programm und haben denselben Datentyp (z.B. String), wie die ausgelesenen Werte der SQL-Datenbank. Die Namensbezeichnung hat ein System welches anhand eines Beispiels kurz erläutert wird.

Die Abkürzung „gbes_zu“ steht für folgende Worte:

- „g“ steht für global
- „bes“ steht für Be-/Entladestation
- „_zu“ steht für das Attribut Zustand

Boolean ist der Variablentyp von „gbes_zu“. Alle Daten der SQL-Datenbank haben einen identischen Datentyp und können so richtig übergeben werden.

Mit Hilfe dieses Unterprogramms werden die Daten aus der SQL-Datenbank ständig abgerufen und angezeigt.

```
Dim gbes_zu As Boolean, gbes_a As Boolean, gbes_bel As Boolean, -
gbes_palid As String, gbes_lotid As String, - usw.
```

Der vollständige Programmcode ist im Anhang A: Programmcode zu finden.

Das Unterprogramm „Private Sub Timer1_Timer()“ wird in einem bestimmten Intervall ausgelöst und beinhaltet folgenden Programmcode:

```
Private Sub Timer1_Timer()

    Set ConnUnt = New ADODB.Connection
    ConnUnt.Provider = "sqloledb.1"
    ConnUnt.Properties("Data Source").Value = "eftpeh"
    ConnUnt.Properties("Initial Catalog").Value = "TestDB"
     ' Datenbank heißt TestDB
    ConnUnt.Properties("Current Language").Value = "German"
    ' con_op.SQL_Language, German
    ConnUnt.Properties("User ID") = "sa"
    ' SQL Username ist sa
    ConnUnt.Properties("Password") = "mes2007"#
    ' SQL Passwort ist mes2007
    ConnUnt.CursorLocation = adUseClient
    ConnUnt.Open

    ssql = "SELECT * FROM MES2 "
    ' Hier wird der SQL-Abfrage String auf die Variable ssql
    ' hinterlegt
    Set rs = New ADODB.Recordset
    rs.Open ssql, ConnUnt, adOpenKeyset, adLockOptimistic
    ' Das Recordset wird geöfnet und mit den Daten aus der Datenbank
    ' gefüllt
```

```
    ' Hier werden alle Daten an die globale Variablen übergeben
    gbes_zu = rs.Fields("gbes_zu")
    gbes_a = rs.Fields("gbes_a") - usw.

    gbearb_zu = rs.Fields("gbearb_zu")
    gbearb_a = rs.Fields("gbearb_a") - usw.

    groh_a = rs.Fields("groh_a")
    groh_zu = rs.Fields("groh_zu") - usw.

    gfertig_a = rs.Fields("gfertig_a")
    gfertig_zu = rs.Fields("gfertig_zu")

    grob_bel = rs.Fields("grob_bel")
    grob_a = rs.Fields("grob_a") - usw.

    gmaschine_bel = rs.Fields("gmaschine_bel")
    gmaschine_a = rs.Fields("gmaschine_a")

    gum1_a = rs.Fields("gum1_a")
    gum2_a = rs.Fields("gum2_a")

    rs.Close
    Set rs = Nothing
    ' Das Recordset wird wieder geschlossen und auf Null zurückgesetzt

    ' Dieser Teil Schliest die Conection wieder und setzt die Variable
auf Nothing
    ConnUnt.Close
    Set ConnUnt = Nothing

    ' Hier werden die Textfelder beschrieben, mit Hilfe der globalen
    ' Variablen die zuvor ausgelesen wurden
    bes_palid.Text = gbes_palid
```

```
bearb_palid.Text = gbearb_palid
fertig_palid.Text = gfertig_palid
roh_palid.Text = groh_palid

bes_lotid.Text = gbes_lotid
bearb_lotid.Text = gbearb_lotid
fertig_lotid.Text = gfertig_lotid
roh_lotid.Text = groh_lotid

bes_fertigmat.Text = gbes_fertigmat
bearb_fertigmat.Text = gbearb_fertigmat
fertig_fertigmat.Text = gfertig_fertigmat
roh_fertigmat.Text = groh_fertigmat

bes_mengefertig.Text = gbes_mengefertig
bearb_mengefertig.Text = gbearb_mengefertig
fertig_mengefertig.Text = gfertig_mengefertig
roh_mengefertig.Text = groh_mengefertig

' Vorerst werden alle MengenTextfelder ausgeblendet
bes_mengeroh.Visible = False
bes_mengefertig.Visible = False - usw.



' Farbe ändern wenn Rohteilpuffer Band aktiv ist
If groh_a = True Then
    roh_a.BackColor = &HFF00&
    roh_palid.BackColor = &HFF00& - usw.

    bes_fertigmat.BackColor = &HFF00&
    roh_mengeroh.BackColor = &HFF00& - usw.

Else
    roh_a.BackColor = &H808080
```

```
        roh_palid.BackColor = &H808080 - usw.

    End If

    ' Farbe ändern wenn Fertigteilpuffer Band aktiv ist
    If gfertig_a = True Then

        fertig_a.BackColor = &HFF00&
        fertig_palid.BackColor = &HFF00& - usw.
    Else
        fertig_a.BackColor = &H808080
        fertig_palid.BackColor = &H808080 - usw.
        bearb_mengefertig.BackColor = &H808080
    End If

    ' Farbe für Umsetzer 1 ändern falls er aktiv ist
    If gum1_a = True Then
        um1_a.BackColor = &HFF00&
    Else
        um1_a.BackColor = &H808080
    End If

    ' Farbe für Umsetzer 2 ändern falls er aktiv ist
    If gum2_a = True Then
        um2_a.BackColor = &HFF00&
    Else
        um2_a.BackColor = &H808080
    End If

    'Zustände oben / unten auf die Test Box der Hubzylinder schreiben
    If gbes_zu = True Then
        bes_zu.Text = "oben"
    Else
        bes_zu.Text = "unten"
```

```
End If

If gbearb_zu = True Then
    bearb_zu.Text = "oben"
Else
    bearb_zu.Text = "unten"
End If

 If gfertig_zu = True Then
    fertig_zu.Text = "oben"
Else
    fertig_zu.Text = "unten"
End If

 If groh_zu = True Then
    roh_zu.Text = "oben"
Else
    roh_zu.Text = "unten"
End If

' Roboter einblenden wenn er aktiv ist
If grob_a = True Then
    'rob.Picture = -
    'LoadPicture("C:\Diplomarbeit_2007_MES\abb_roboter2.jpg")
    rob.Visible = True

Else
    rob.Visible = False
End If

' Maschine einblenden wenn sie aktiv ist
If gmaschine_a = True Then
    'rob.Picture = -
    'LoadPicture("C:\Diplomarbeit_2007_MES\abb_roboter2.jpg")
```

```
        maschine.Visible = True

    Else
        maschine.Visible = False
    End If

    ' Farbe der Paletten ändern wenn eine vorhanden ist
    If bes_palid.Text <> "" Then

        bes_mengeroh.Visible = True
        bes_mengefertig.Visible = True

        bes_palid.BackColor = &H80FFFF
        bes_lotid.BackColor = &H80FFFF - usw.
    End If

    If bearb_palid.Text <> "" Then

        bearb_mengeroh.Visible = True
        bearb_mengefertig.Visible = True

        bearb_palid.BackColor = &H80FFFF
        bearb_lotid.BackColor = &H80FFFF - usw.
    End If

    If fertig_palid.Text <> "" Then

        fertig_mengeroh.Visible = True
        fertig_mengefertig.Visible = True
        fertig_palid.BackColor = &H80FFFF
        fertig_lotid.BackColor = &H80FFFF - usw.
    End If

    If roh_palid.Text <> "" Then
```

```
        roh_mengeroh.Visible = True
        roh_mengefertig.Visible = True
        roh_palid.BackColor = &H80FFFF
        roh_lotid.BackColor = &H80FFFF - usw.
    End If
End Sub
```

Der Simulator ist völlig unabhängig von der Bedieneroberfläche der Anlage und enthält keine CAB Elemente. Er kann daher auf jedem PC im Netzwerk gestartet werden, der einen Zugriff auf den SQL Server hat auf welchen die Datenbank liegt.

12 Zusammenfassung und Ausblick

Der Einsatz von Manufacturing Execution Systems in der diskreten Fertigung ist für viele Unternehmen ein aktuelles Thema. Ziel dieser Diplomarbeit war es deshalb, eine Basis für Forschungsarbeiten zum Thema Manufacturing Execution Systems zu bilden. Aus diesem Grund wurde eine Fertigungszelle modelliert und simuliert. Dazu wurde die MES-Software „SIMATIC IT Production Suite Version 6“ der Firma SIEMENS verwendet. Es wurde ein Grundaufbau mit wesentlichen Funktionen eines MES realisiert und es wurde dabei wie folgt vorgegangen.

Zuerst wurde die Versuchsanlage im Production Modeler von SIMATIC IT erstellt. Dabei wurden alle Anlagenkomponenten und Prozesse, die in der Anlage ablaufen, modelliert.

Nach dem Modellieren sollten die abgebildeten Komponenten und Prozesse über eine Schnittstelle (COM-Wrapper) mit der Anlage verbunden werden. Da die Anlage aber physisch nicht zur Verfügung gestanden ist, wurde sie durch einen Simulator ersetzt, der alle Komponenten der realen Anlage darstellt. Der COM-Wrapper und der Simulator wurden in Microsoft Visual Basic programmiert.

Als nächstes wurde die Verbindung zwischen dem Manufacturing Execution System und einem Enterprise Ressource Planning System realisiert. Dies war notwendig, um Aufträge aus dem ERP ins MES übernehmen zu können. Der Datenaustausch erfolgte über XML-Dateien, welche eine genau definierte Struktur besitzen mussten.

Um eine Anlagenbedienung zu ermöglichen, musste noch eine Benutzeroberfläche für den Bedien-PC der Anlage erstellt werden. Dies geschah mit Hilfe der Software MS Visual Basic.net und einem Plug-In (CAB), das von SIMATIC IT Production Suite zur Verfügung gestellt wurde. Somit wurden alle Funktionalitäten der Anlage sowie die Schnittstellen zu anderen Ebenen geschaffen.

Während der Erstellung der Diplomarbeit konnten sehr viele Erfahrungen über den Einsatz von Manufacturing Execution Systems gewonnen werden. Es wurden die Einsatzmöglichkeiten der verschiedenen Komponenten, deren Zusammenspiel und die Verbindungen zu anderen Systemen verdeutlicht.

Mit dieser Diplomarbeit wurde ein Grundstein für weiterführende Forschungsarbeiten gelegt. Es ist nun eine Basis vorhanden, auf der weitere Arbeiten aufsetzen können. Diese könnten das Implementieren anderer MES-Komponenten, die Anbindung an die reale Fertigungszelle sowie das Erweitern der Versuchsanlage umfassen.

Anhang A: Programmcode

In Anhang A findet man den vollständigen Programmcode für den Simulator und für alle Methoden der DLL. Beide Programme sind in MS Visual Basic 6.0 programmiert worden.

A.1 DLL Code

```
Dim ssql As String
Dim rs As ADODB.Recordset
Dim strStream As ADODB.Stream
Dim x As Variant
Dim i As Double
Dim ConnUnt As ADODB.Connection

Dim gbes_zu As Boolean, gbes_a As Boolean, gbes_bel As Boolean, -
gbes_palid As String, gbes_lotid As String, -
gbes_fertigmat As String, gbes_rohmat As String, -
gbes_mengeroh As Integer, gbes_mengefertig As Integer

Dim groh_a As Boolean, groh_zu As Boolean, groh_bel As Boolean, -
groh_palid As String, groh_lotid As String, -
groh_fertigmat As String, groh_rohmat As String, -
groh_mengeroh As Integer, groh_mengefertig As Integer

Dim gum1_a As Boolean, gum2_a As Boolean
Dim gfertig_a As Boolean, gfertig_zu As Boolean, gfertig_bel As -
Boolean, gfertig_palid As String, gfertig_lotid As String, -
gfertig_fertigmat As String, gfertig_rohmat As String, -
gfertig_mengeroh As Integer, gfertig_mengefertig As Integer

Dim gbearb_zu As Boolean, gbearb_a As Boolean, -
gbearb_bel As Boolean, gbearb_palid As String, -
```

```
gbearb_lotid As String, gbearb_fertigmat As String, -
gbearb_rohmat As String, gbearb_mengeroh As Integer, -
gbearb_mengefertig As Integer

Dim gmaschine_a As Boolean, gmaschine_bel As Boolean, grob_a As -
Boolean, grob_bel As Boolean

Public Sub Initialisieren()

    gbes_zu = True
    gbes_a = False
    gbes_bel = False
    gbes_palid = ""
    gbes_lotid = ""
    gbes_fertigmat = ""
    gbes_rohmat = ""
    gbes_mengeroh = 0
    gbes_mengefertig = 0

    gbearb_zu = True
    gbearb_a = False
    gbearb_bel = False
    gbearb_palid = ""
    gbearb_lotid = ""
    gbearb_fertigmat = ""
    gbearb_rohmat = ""
    gbearb_mengeroh = 0
    gbearb_mengefertig = 0

    groh_a = False
    groh_zu = True
    groh_bel = False
    groh_palid = ""
    groh_lotid = ""
```

```
groh_fertigmat = ""
groh_rohmat = ""
groh_mengeroh = 0
groh_mengefertig = 0

gfertig_a = False
gfertig_zu = True
gfertig_bel = False
gfertig_palid = ""
gfertig_lotid = ""
gfertig_fertigmat = ""
gfertig_rohmat = ""
gfertig_mengeroh = 0
gfertig_mengefertig = 0

grob_bel = False
grob_a = False

gmaschine_bel = False
gmaschine_a = False

gum1_a = False
gum2_a = False

Set ConnUnt = New ADODB.Connection
ConnUnt.Provider = "sqloledb.1"
ConnUnt.Properties("Data Source").Value = "eftpeh"
ConnUnt.Properties("Initial Catalog").Value = "TestDB"
' Datenbank
ConnUnt.Properties("Current Language").Value = "German"
'con_op.SQL_Language  ' German

ConnUnt.Properties("User ID") = "sa"
' SQL Username
```

```
ConnUnt.Properties("Password") = "mes2007"
' SQL Passwort
ConnUnt.CursorLocation = adUseClient

ConnUnt.Open

ssql = "SELECT * FROM MES2 "
Set rs = New ADODB.Recordset
rs.Open ssql, ConnUnt, adOpenKeyset, adLockOptimistic

rs.Fields("gbes_zu") = gbes_zu
rs.Fields("gbes_a") = gbes_a
rs.Fields("gbes_bel") = gbes_bel
rs.Fields("gbes_palid") = gbes_palid
rs.Fields("gbes_lotid") = gbes_lotid
rs.Fields("gbes_fertigmat") = gbes_fertigmat
rs.Fields("gbes_rohmat") = gbes_rohmat
rs.Fields("gbes_mengeroh") = gbes_mengeroh
rs.Fields("gbes_mengefertig") = gbes_mengefertig

rs.Fields("gbearb_zu") = gbearb_zu
rs.Fields("gbearb_a") = gbearb_a
rs.Fields("gbearb_bel") = gbearb_bel
rs.Fields("gbearb_palid") = gbearb_palid
rs.Fields("gbearb_lotid") = gbearb_lotid
rs.Fields("gbearb_fertigmat") = gbearb_fertigmat
rs.Fields("gbearb_rohmat") = gbearb_rohmat
rs.Fields("gbearb_mengeroh") = gbearb_mengeroh
rs.Fields("gbearb_mengefertig") = gbearb_mengefertig

rs.Fields("groh_a") = groh_a
rs.Fields("groh_zu") = groh_zu
rs.Fields("groh_bel") = groh_bel
rs.Fields("groh_palid") = groh_palid
```

```
    rs.Fields("groh_lotid") = groh_lotid
    rs.Fields("groh_fertigmat") = groh_fertigmat
    rs.Fields("groh_rohmat") = groh_rohmat
    rs.Fields("groh_mengeroh") = groh_mengeroh
    rs.Fields("groh_mengefertig") = groh_mengefertig

    rs.Fields("gfertig_a") = gfertig_a
    rs.Fields("gfertig_zu") = gfertig_zu
    rs.Fields("gfertig_bel") = gfertig_bel
    rs.Fields("gfertig_palid") = gfertig_pali
    rs.Fields("gfertig_lotid") = gfertig_lotid
    rs.Fields("gfertig_fertigmat") = gfertig_fertigmat
    rs.Fields("gfertig_rohmat") = gfertig_rohmat
    rs.Fields("gfertig_mengeroh") = gfertig_mengeroh
    rs.Fields("gfertig_mengefertig") = gfertig_mengefertig

    rs.Fields("grob_bel") = grob_bel
    rs.Fields("grob_a") = grob_a

    rs.Fields("gmaschine_bel") = gmaschine_bel
    rs.Fields("gmaschine_a") = gmaschine_a

    rs.Fields("gum1_a") = gum1_a
    rs.Fields("gum2_a") = gum2_a

    rs.Update
    rs.Close

    Set rs = Nothing

    ConnUnt.Close
    Set ConnUnt = Nothing

End Sub
```

```
Public Sub Zustandpr(bes_zu As Boolean, bes_a As Boolean,-
bes_bel As Boolean, bes_palid As String, roh_a As Boolean, -
roh_zu As Boolean, roh_bel As Boolean, roh_palid As String, -
um1_a As Boolean, um2_a As Boolean, fertig_a As Boolean, -
fertig_zu As Boolean, fertig_bel As Boolean, fertig_palid As String, -
bearb_zu As Boolean, bearb_a As Boolean, bearb_bel As Boolean, -
bearb_palid As String, bearb_lotid As String, -
bearb_fertigmat As String, bearb_rohmat As String, -
maschine_a As Boolean, maschine_bel As Boolean, rob_a As Boolean, -
rob_bel As Boolean)

    Set ConnUnt = New ADODB.Connection
    ConnUnt.Provider = "sqloledb.1"
    ConnUnt.Properties("Data Source").Value = "eftpeh"
    ConnUnt.Properties("Initial Catalog").Value = "TestDB"
    ' Datenbank
    ConnUnt.Properties("Current Language").Value = "German"
    'con_op.SQL_Language  ' German

    ConnUnt.Properties("User ID") = "sa"
    ' SQL Username
    ConnUnt.Properties("Password") = "mes2007"
    ' SQL Passwort
    ConnUnt.CursorLocation = adUseClient

    ConnUnt.Open

    ssql = "SELECT * FROM MES2 "
    Set rs = New ADODB.Recordset
    rs.Open ssql, ConnUnt, adOpenKeyset, adLockOptimistic

    bes_zu = rs.Fields("gbes_zu")
```

```
bes_a = rs.Fields("gbes_a")
bes_bel = rs.Fields("gbes_bel")
bes_palid = rs.Fields("gbes_palid")

bearb_zu = rs.Fields("gbearb_zu")
bearb_a = rs.Fields("gbearb_a")
bearb_bel = rs.Fields("gbearb_bel")
bearb_palid = rs.Fields("gbearb_palid")
bearb_lotid = rs.Fields("gbearb_lotid")
bearb_fertigmat = rs.Fields("gbearb_fertigmat")
bearb_rohmat = rs.Fields("gbearb_rohmat")

roh_a = rs.Fields("groh_a")
roh_zu = rs.Fields("groh_zu")
roh_bel = rs.Fields("groh_bel")
roh_palid = rs.Fields("groh_palid")

fertig_a = rs.Fields("gfertig_a")
fertig_zu = rs.Fields("gfertig_zu")
fertig_bel = rs.Fields("gfertig_bel")
fertig_palid = rs.Fields("gfertig_palid")

rob_bel = rs.Fields("grob_bel")
rob_a = rs.Fields("grob_a")

maschine_bel = rs.Fields("gmaschine_bel")
maschine_a = rs.Fields("gmaschine_a")

um1_a = rs.Fields("gum1_a")
um2_a = rs.Fields("gum2_a")

rs.Close

Set rs = Nothing
```

```
    ConnUnt.Close
    Set ConnUnt = Nothing

End Sub



Public Sub Zustandsetz(bes_zu As Boolean, bes_a As Boolean, -
bes_bel As Boolean, bes_palid As String, roh_a As Boolean, -
roh_zu As Boolean, roh_bel As Boolean, roh_palid As String, -
um1_a As Boolean, um2_a As Boolean, fertig_a As Boolean, -
fertig_zu As Boolean, fertig_bel As Boolean, fertig_palid As String, -
bearb_zu As Boolean, bearb_a As Boolean, bearb_bel As Boolean, -
bearb_palid As String, maschine_a As Boolean, -
maschine_bel As Boolean, rob_a As Boolean, rob_bel As Boolean)

    Set ConnUnt = New ADODB.Connection
    ConnUnt.Provider = "sqloledb.1"
    ConnUnt.Properties("Data Source").Value = "eftpeh"
    ConnUnt.Properties("Initial Catalog").Value = "TestDB"
    ' Datenbank
    ConnUnt.Properties("Current Language").Value = "German"
    'con_op.SQL_Language  ' German

    ConnUnt.Properties("User ID") = "sa"
    ' SQL Username
    ConnUnt.Properties("Password") = "mes2007"
    ' SQL Passwort
    ConnUnt.CursorLocation = adUseClient

    ConnUnt.Open

    ssql = "SELECT * FROM MES2 "
```

```
Set rs = New ADODB.Recordset
rs.Open ssql, ConnUnt, adOpenKeyset, adLockOptimistic

rs.Fields("gbes_zu") = bes_zu
rs.Fields("gbes_a") = bes_a
rs.Fields("gbes_bel") = bes_bel
rs.Fields("gbes_palid") = bes_palid

rs.Fields("gbearb_zu") = bearb_zu
rs.Fields("gbearb_a") = bearb_a
rs.Fields("gbearb_bel") = bearb_bel
rs.Fields("gbearb_palid") = bearb_palid

rs.Fields("groh_a") = roh_a
rs.Fields("groh_zu") = roh_zu
rs.Fields("groh_bel") = roh_bel
rs.Fields("groh_palid") = roh_palid

rs.Fields("gfertig_a") = fertig_a
rs.Fields("gfertig_zu") = fertig_zu
rs.Fields("gfertig_bel") = fertig_bel
rs.Fields("gfertig_palid") = fertig_pali

rs.Fields("grob_bel") = rob_bel
rs.Fields("grob_a") = rob_a

rs.Fields("gmaschine_bel") = maschine_bel
rs.Fields("gmaschine_a") = maschine_a

rs.Fields("gum1_a") = um1_a
rs.Fields("gum2_a") = um2_a

rs.Close
```

```
    Set rs = Nothing

    ConnUnt.Close
    Set ConnUnt = Nothing

End Sub



Public Sub palbestuecken(bes_palid As String, bes_lotid As String, -
bes_rohmat As String, bes_fertigmat As String, -
bes_mengeroh As Integer)

    Set ConnUnt = New ADODB.Connection
    ConnUnt.Provider = "sqloledb.1"
    ConnUnt.Properties("Data Source").Value = "eftpeh"
    ConnUnt.Properties("Initial Catalog").Value = "TestDB"
    ' Datenbank
    ConnUnt.Properties("Current Language").Value = "German"
    'con_op.SQL_Language  ' German

    ConnUnt.Properties("User ID") = "sa"
    ' SQL Username
    ConnUnt.Properties("Password") = "mes2007"
    ' SQL Passwort
    ConnUnt.CursorLocation = adUseClient

    ConnUnt.Open

    ssql = "SELECT * FROM MES2 "
    Set rs = New ADODB.Recordset
    rs.Open ssql, ConnUnt, adOpenKeyset, adLockOptimistic

    gbes_a = True
```

```
    rs.Fields("gbes_a") = gbes_a

    rs.Fields("gbes_palid") = bes_palid

    rs.Fields("gbes_lotid") = bes_lotid

    rs.Fields("gbes_fertigmat") = bes_fertigmat

    rs.Fields("gbes_rohmat") = bes_rohmat

    rs.Fields("gbes_mengeroh") = bes_mengeroh

    rs.Fields("gbes_mengefertig") = 0

    rs.Update
    'um den AKTIV Zusatand in der DB zu aktualisieren

    MsgBox "Bestuecken abgeschlossen?"   'statt Delay

    gbes_bel = True
    rs.Fields("gbes_bel") = gbes_bel

    gbes_a = False
    rs.Fields("gbes_a") = gbes_a

    rs.Update
    rs.Close

    Set rs = Nothing

    ConnUnt.Close
    Set ConnUnt = Nothing

End Sub
```

```
Public Sub palentnehmen()

    Set ConnUnt = New ADODB.Connection
    ConnUnt.Provider = "sqloledb.1"
    ConnUnt.Properties("Data Source").Value = "eftpeh"
    ConnUnt.Properties("Initial Catalog").Value = "TestDB"
    ' Datenbank
    ConnUnt.Properties("Current Language").Value = "German"
    'con_op.SQL_Language  ' German

    ConnUnt.Properties("User ID") = "sa"
    ' SQL Username
    ConnUnt.Properties("Password") = "mes2007"
    ' SQL Passwort
    ConnUnt.CursorLocation = adUseClient

    ConnUnt.Open

    ssql = "SELECT * FROM MES2 " '& "WHERE KITTL = 6"
    Set rs = New ADODB.Recordset
    rs.Open ssql, ConnUnt, adOpenKeyset, adLockOptimistic

    gbes_a = True
    rs.Fields("gbes_a") = gbes_a

    rs.Update
    'um den AKTIV Zusatand in der DB zu aktualisieren

    MsgBox "Entnehmen abgeschlossen?"   'statt Delay

    'bes_palid = gbes_palid
    gbes_palid = ""
```

```
    rs.Fields("gbes_palid") = gbes_palid

    gbes_lotid = ""
    rs.Fields("gbes_lotid") = gbes_lotid

    gbes_fertigmat = ""
    rs.Fields("gbes_fertigmat") = gbes_fertigmat

    gbes_rohmat = ""
    rs.Fields("gbes_rohmat") = gbes_rohmat

    gbes_mengeroh = 0
    rs.Fields("gbes_mengeroh") = gbes_mengeroh

    gbes_mengefertig = 0
    rs.Fields("gbes_mengefertig") = gbes_mengefertig

    gbes_a = False
    rs.Fields("gbes_a") = gbes_a

    gbes_bel = False
    rs.Fields("gbes_bel") = gbes_bel

    rs.Update
    rs.Close

    Set rs = Nothing

    ConnUnt.Close
    Set ConnUnt = Nothing

End Sub
```

```
Public Sub beszupuffer()

Set ConnUnt = New ADODB.Connection
    ConnUnt.Provider = "sqloledb.1"
    ConnUnt.Properties("Data Source").Value = "eftpeh"
    ConnUnt.Properties("Initial Catalog").Value = "TestDB"
    ' Datenbank
    ConnUnt.Properties("Current Language").Value = "German"
    'con_op.SQL_Language  ' German

    ConnUnt.Properties("User ID") = "sa"
    ' SQL Username
    ConnUnt.Properties("Password") = "mes2007"
    ' SQL Passwort
    ConnUnt.CursorLocation = adUseClient

    ConnUnt.Open

    ssql = "SELECT * FROM MES2 "
    Set rs = New ADODB.Recordset
    rs.Open ssql, ConnUnt, adOpenKeyset, adLockOptimistic

    gbes_zu = False
    rs.Fields("gbes_zu") = gbes_zu

    groh_zu = False
    rs.Fields("groh_zu") = groh_zu

    groh_a = True
    rs.Fields("groh_a") = groh_a

    gbes_a = True
    rs.Fields("gbes_a") = gbes_a
```

```
rs.Update
'um den AKTIV Zusatand in der DB zu aktualisieren

MsgBox "BES zu Rohteilpuffer."    'statt Delay

gbes_palid = rs.Fields("gbes_palid")
groh_palid = gbes_palid
gbes_palid = ""
rs.Fields("gbes_palid") = gbes_palid
rs.Fields("groh_palid") = groh_palid

gbes_lotid = rs.Fields("gbes_lotid")
groh_lotid = gbes_lotid
gbes_lotid = ""
rs.Fields("gbes_lotid") = gbes_lotid
rs.Fields("groh_lotid") = groh_lotid

gbes_rohmat = rs.Fields("gbes_rohmat")
groh_rohmat = gbes_rohmat
gbes_rohmat = ""
rs.Fields("gbes_rohmat") = gbes_rohmat
rs.Fields("groh_rohmat") = groh_rohmat

gbes_fertigmat = rs.Fields("gbes_fertigmat")
groh_fertigmat = gbes_fertigmat
gbes_fertigmat = ""
rs.Fields("gbes_fertigmat") = gbes_fertigmat
rs.Fields("groh_fertigmat") = groh_fertigmat

gbes_mengefertig = rs.Fields("gbes_mengefertig")
groh_mengefertig = gbes_mengefertig
gbes_mengefertig = 0
rs.Fields("gbes_mengefertig") = gbes_mengefertig
```

```
    rs.Fields("groh_mengefertig") = groh_mengefertig

    gbes_mengeroh = rs.Fields("gbes_mengeroh")
    groh_mengeroh = gbes_mengeroh
    gbes_mengeroh = 0
    rs.Fields("gbes_mengeroh") = gbes_mengeroh
    rs.Fields("groh_mengeroh") = groh_mengeroh

    gbes_bel = False
    rs.Fields("gbes_bel") = gbes_bel

    groh_bel = True
    rs.Fields("groh_bel") = groh_bel

    groh_a = False
    rs.Fields("groh_a") = groh_a

    gbes_a = False
    rs.Fields("gbes_a") = gbes_a

    gbes_zu = True
    rs.Fields("gbes_zu") = gbes_zu

    groh_zu = True
    rs.Fields("groh_zu") = groh_zu

    rs.Update
    rs.Close

    Set rs = Nothing

    ConnUnt.Close
    Set ConnUnt = Nothing
End Sub
```

```
Public Sub pufferzubearb()

Set ConnUnt = New ADODB.Connection
    ConnUnt.Provider = "sqloledb.1"
    ConnUnt.Properties("Data Source").Value = "eftpeh"
    ConnUnt.Properties("Initial Catalog").Value = "TestDB"
    ' Datenbank
    ConnUnt.Properties("Current Language").Value = "German"
    'con_op.SQL_Language  ' German

    ConnUnt.Properties("User ID") = "sa"
    ' SQL Username
    ConnUnt.Properties("Password") = "mes2007"
    ' SQL Passwort
    ConnUnt.CursorLocation = adUseClient

    ConnUnt.Open

    ssql = "SELECT * FROM MES2 " '& "WHERE KITTL = 6"
    Set rs = New ADODB.Recordset
    rs.Open ssql, ConnUnt, adOpenKeyset, adLockOptimistic

    groh_a = True
    rs.Fields("groh_a") = groh_a

    gum1_a = True
    rs.Fields("gum1_a") = gum1_a

    gfertig_a = True
    rs.Fields("gfertig_a") = gfertig_a

    gbearb_zu = False
```

```
rs.Fields("gbearb_zu") = gbearb_zu

groh_zu = False
rs.Fields("groh_zu") = groh_zu

rs.Update
'um den AKTIV Zusatand in der DB zu aktualisieren

MsgBox "Rohteilpuffer zu Bearb_Station."         'statt Delay

groh_palid = rs.Fields("groh_palid")
gbearb_palid = groh_palid
groh_palid = ""
rs.Fields("gbearb_palid") = gbearb_palid
rs.Fields("groh_palid") = groh_palid

groh_lotid = rs.Fields("groh_lotid")
gbearb_lotid = groh_lotid
groh_lotid = ""
rs.Fields("gbearb_lotid") = gbearb_lotid
rs.Fields("groh_lotid") = groh_lotid

groh_rohmat = rs.Fields("groh_rohmat")
gbearb_rohmat = groh_rohmat
groh_rohmat = ""
rs.Fields("gbearb_rohmat") = gbearb_rohmat
rs.Fields("groh_rohmat") = groh_rohmat

groh_fertigmat = rs.Fields("groh_fertigmat")
gbearb_fertigmat = groh_fertigmat
groh_fertigmat = ""
rs.Fields("gbearb_fertigmat") = gbearb_fertigmat
rs.Fields("groh_fertigmat") = groh_fertigmat
```

```
groh_mengefertig = rs.Fields("groh_mengefertig")
gbearb_mengefertig = groh_mengefertig
groh_mengefertig = 0
rs.Fields("gbearb_mengefertig") = gbearb_mengefertig
rs.Fields("groh_mengefertig") = groh_mengefertig

groh_mengeroh = rs.Fields("groh_mengeroh")
gbearb_mengeroh = groh_mengeroh
groh_mengeroh = 0
rs.Fields("gbearb_mengeroh") = gbearb_mengeroh
rs.Fields("groh_mengeroh") = groh_mengeroh

groh_a = False
rs.Fields("groh_a") = groh_a

gum1_a = False
rs.Fields("gum1_a") = gum1_a

gfertig_a = False
rs.Fields("gfertig_a") = gfertig_a

gbearb_zu = True
rs.Fields("gbearb_zu") = gbearb_zu

groh_zu = True
rs.Fields("groh_zu") = groh_zu

groh_bel = False
rs.Fields("groh_bel") = groh_bel

gbearb_bel = True
rs.Fields("gbearb_bel") = gbearb_bel

rs.Update
```

```
    rs.Close
    Set rs = Nothing

    ConnUnt.Close
    Set ConnUnt = Nothing

End Sub



Public Sub bearbzupuffer()

Set ConnUnt = New ADODB.Connection
    ConnUnt.Provider = "sqloledb.1"
    ConnUnt.Properties("Data Source").Value = "eftpeh"
    ConnUnt.Properties("Initial Catalog").Value = "TestDB"
    ' Datenbank
    ConnUnt.Properties("Current Language").Value = "German"
    'con_op.SQL_Language  ' German

    ConnUnt.Properties("User ID") = "sa"
    ' SQL Username
    ConnUnt.Properties("Password") = "mes2007"
    ' SQL Passwort
    ConnUnt.CursorLocation = adUseClient

    ConnUnt.Open

    ssql = "SELECT * FROM MES2 "
    Set rs = New ADODB.Recordset
    rs.Open ssql, ConnUnt, adOpenKeyset, adLockOptimistic

    gfertig_a = True
    rs.Fields("gfertig_a") = gfertig_a
```

```
gbearb_zu = False
rs.Fields("gbearb_zu") = gbearb_zu

gfertig_zu = False
rs.Fields("gfertig_zu") = gfertig_zu

rs.Update
'um den AKTIV Zusatand in der DB zu aktualisieren

MsgBox "BEARB zu Fertigteilpuffer."        'statt Delay

gbearb_palid = rs.Fields("gbearb_palid")
gfertig_palid = gbearb_palid
gbearb_palid = ""
rs.Fields("gbearb_palid") = gbearb_palid
rs.Fields("gfertig_palid") = gfertig_palid

gbearb_lotid = rs.Fields("gbearb_lotid")
gfertig_lotid = gbearb_lotid
gbearb_lotid = ""
rs.Fields("gbearb_lotid") = gbearb_lotid
rs.Fields("gfertig_lotid") = gfertig_lotid

gbearb_rohmat = rs.Fields("gbearb_rohmat")
gfertig_rohmat = gbearb_rohmat
gbearb_rohmat = ""
rs.Fields("gbearb_rohmat") = gbearb_rohmat
rs.Fields("gfertig_rohmat") = gfertig_rohmat

gbearb_fertigmat = rs.Fields("gbearb_fertigmat")
gfertig_fertigmat = gbearb_fertigmat
gbearb_fertigmat = ""
rs.Fields("gbearb_fertigmat") = gbearb_fertigmat
```

```
rs.Fields("gfertig_fertigmat") = gfertig_fertigmat

gbearb_mengefertig = rs.Fields("gbearb_mengefertig")
gfertig_mengefertig = gbearb_mengefertig
gbearb_mengefertig = 0
rs.Fields("gbearb_mengefertig") = gbearb_mengefertig
rs.Fields("gfertig_mengefertig") = gfertig_mengefertig

gbearb_mengeroh = rs.Fields("gbearb_mengeroh")
gfertig_mengeroh = gbearb_mengeroh
gbearb_mengeroh = 0
rs.Fields("gbearb_mengeroh") = gbearb_mengeroh
rs.Fields("gfertig_mengeroh") = gfertig_mengeroh

gfertig_a = False
rs.Fields("gfertig_a") = gfertig_a

gbearb_zu = True
rs.Fields("gbearb_zu") = gbearb_zu

gfertig_zu = True
rs.Fields("gfertig_zu") = gfertig_zu

gbearb_bel = False
rs.Fields("gbearb_bel") = gbearb_bel

gfertig_bel = True
rs.Fields("gfertig_bel") = gfertig_bel

rs.Update
rs.Close

Set rs = Nothing
```

```
    ConnUnt.Close
    Set ConnUnt = Nothing

End Sub



Public Sub pufferzubes()

Set ConnUnt = New ADODB.Connection
    ConnUnt.Provider = "sqloledb.1"
    ConnUnt.Properties("Data Source").Value = "eftpeh"
    ConnUnt.Properties("Initial Catalog").Value = "TestDB"
    ' Datenbank
    ConnUnt.Properties("Current Language").Value = "German"
    'con_op.SQL_Language  ' German

    ConnUnt.Properties("User ID") = "sa"
    ' SQL Username
    ConnUnt.Properties("Password") = "mes2007"
    ' SQL Passwort
    ConnUnt.CursorLocation = adUseClient

    ConnUnt.Open

    ssql = "SELECT * FROM MES2 "
    Set rs = New ADODB.Recordset
    rs.Open ssql, ConnUnt, adOpenKeyset, adLockOptimistic

    gfertig_a = True
    rs.Fields("gfertig_a") = gfertig_a

    groh_a = True
    rs.Fields("groh_a") = groh_a
```

```
gum2_a = True
rs.Fields("gum2_a") = gum2_a

gbes_zu = False
rs.Fields("gbes_zu") = gbes_zu

gfertig_zu = False
rs.Fields("gfertig_zu") = gfertig_zu

rs.Update
'um den AKTIV Zusatand in der DB zu aktualisieren

MsgBox "Fertigteilpuffer zu BES_Station."   'statt Delay

gfertig_palid = rs.Fields("gfertig_palid")
gbes_palid = gfertig_palid
gfertig_palid = ""
rs.Fields("gbes_palid") = gbes_palid
rs.Fields("gfertig_palid") = gfertig_palid

gfertig_lotid = rs.Fields("gfertig_lotid")
gbes_lotid = gfertig_lotid
gfertig_lotid = ""
rs.Fields("gbes_lotid") = gbes_lotid
rs.Fields("gfertig_lotid") = gfertig_lotid

gfertig_rohmat = rs.Fields("gfertig_rohmat")
gbes_rohmat = gfertig_rohmat
gfertig_rohmat = ""
rs.Fields("gbes_rohmat") = gbes_rohmat
rs.Fields("gfertig_rohmat") = gfertig_rohmat

gfertig_fertigmat = rs.Fields("gfertig_fertigmat")
```

```
gbes_fertigmat = gfertig_fertigmat
gfertig_fertigmat = ""
rs.Fields("gbes_fertigmat") = gbes_fertigmat
rs.Fields("gfertig_fertigmat") = gfertig_fertigmat

gfertig_mengefertig = rs.Fields("gfertig_mengefertig")
gbes_mengefertig = gfertig_mengefertig
gfertig_mengefertig = 0
rs.Fields("gbes_mengefertig") = gbes_mengefertig
rs.Fields("gfertig_mengefertig") = gfertig_mengefertig

gfertig_mengeroh = rs.Fields("gfertig_mengeroh")
gbes_mengeroh = gfertig_mengeroh
gfertig_mengeroh = 0
rs.Fields("gbes_mengeroh") = gbes_mengeroh
rs.Fields("gfertig_mengeroh") = gfertig_mengeroh

gfertig_a = False
rs.Fields("gfertig_a") = gfertig_a

groh_a = False
rs.Fields("groh_a") = groh_a

gum2_a = False
rs.Fields("gum2_a") = gum2_a

gbes_zu = True
rs.Fields("gbes_zu") = gbes_zu

gfertig_zu = True
rs.Fields("gfertig_zu") = gfertig_zu

gfertig_bel = False
rs.Fields("gfertig_bel") = gfertig_bel
```

```
    gbes_bel = True
    rs.Fields("gbes_bel") = gbes_bel

    rs.Update
    rs.Close

    Set rs = Nothing

    ConnUnt.Close
    Set ConnUnt = Nothing

End Sub



Public Sub fertigen()

Set ConnUnt = New ADODB.Connection
    ConnUnt.Provider = "sqloledb.1"
    ConnUnt.Properties("Data Source").Value = "eftpeh"
    ConnUnt.Properties("Initial Catalog").Value = "TestDB"
    ' Datenbank
    ConnUnt.Properties("Current Language").Value = "German"
    'con_op.SQL_Language  ' German

    ConnUnt.Properties("User ID") = "sa"
    ' SQL Username
    ConnUnt.Properties("Password") = "mes2007"
    ' SQL Passwort
    ConnUnt.CursorLocation = adUseClient

    ConnUnt.Open
```

```
    ssql = "SELECT * FROM MES2 "
    Set rs = New ADODB.Recordset
    rs.Open ssql, ConnUnt, adOpenKeyset, adLockOptimistic

    grob_a = True
    rs.Fields("grob_a") = grob_a

    gmaschine_a = True
    rs.Fields("gmaschine_a") = gmaschine_a

    rs.Update
    'um den AKTIV Zusatand in der DB zu aktualisieren

    MsgBox "Einzelstück gefertigt."   'statt Delay

    gbearb_mengeroh = gbearb_mengeroh - 1
    gbearb_mengefertig = gbearb_mengefertig + 1
    rs.Fields("gbearb_mengeroh") = gbearb_mengeroh
    rs.Fields("gbearb_mengefertig") = gbearb_mengefertig

    grob_a = False
    rs.Fields("grob_a") = grob_a

    gmaschine_a = False
    rs.Fields("gmaschine_a") = gmaschine_a

    rs.Update
    rs.Close

    Set rs = Nothing

    ConnUnt.Close
    Set ConnUnt = Nothing
End Sub
```

```
Public Sub fertiglotid(neu_lotid As String)

Set ConnUnt = New ADODB.Connection
    ConnUnt.Provider = "sqloledb.1"
    ConnUnt.Properties("Data Source").Value = "eftpeh"
    ConnUnt.Properties("Initial Catalog").Value = "TestDB"
    ' Datenbank
    ConnUnt.Properties("Current Language").Value = "German"
    'con_op.SQL_Language  ' German

    ConnUnt.Properties("User ID") = "sa"
    ' SQL Username
    ConnUnt.Properties("Password") = "mes2007"
    ' SQL Passwort
    ConnUnt.CursorLocation = adUseClient

    ConnUnt.Open

    ssql = "SELECT * FROM MES2 "
    Set rs = New ADODB.Recordset
    rs.Open ssql, ConnUnt, adOpenKeyset, adLockOptimistic

    rs.Fields("gbearb_lotid") = neu_lotid

    rs.Update
    rs.Close

    Set rs = Nothing

    ConnUnt.Close
    Set ConnUnt = Nothing
End Sub
```

```
Public Sub sucherohmaterial(rohmat As String, fertigmat As String)

    Dim DefIDroh, DefIDfertig As String
    Dim DefPKfertig, BomPK, DefPKroh As Integer

    Set ConnUnt = New ADODB.Connection
    ConnUnt.Provider = "sqloledb.1"
    ConnUnt.Properties("Data Source").Value = "eftpeh"
    ConnUnt.Properties("Initial Catalog").Value = "SITMesDB"
    ' Datenbank
    ConnUnt.Properties("Current Language").Value = "German"
    'con_op.SQL_Language  ' German

    ConnUnt.Properties("User ID") = "sa"
    ' SQL Username
    ConnUnt.Properties("Password") = "mes2007"
    ' SQL Passwort
    ConnUnt.CursorLocation = adUseClient

    ConnUnt.Open

    ssql = "SELECT DefPK,DefID FROM MMDefinitions WHERE -
    DefID = '" & fertigmat & "'"
    Set rs = New ADODB.Recordset
    rs.Open ssql, ConnUnt, adOpenKeyset, adLockOptimistic

    If rs.RecordCount = 0 Then
        MsgBox "Fertigmaterial ist nicht definiert!"
        rs.Close
    Else
        DefPKfertig = rs.Fields("DefPK")
        rs.Close
```

```
        ssql = "SELECT DefPK,BomPK FROM MMBoms WHERE -
        DefPK = '" & DefPKfertig & "'"
        Set rs = New ADODB.Recordset
        rs.Open ssql, ConnUnt, adOpenKeyset, adLockOptimistic

        BomPK = rs.Fields("BomPK")

        rs.Close

        ssql = "SELECT DefPK,BomAltPK FROM MMBomItemAlts WHERE -
        BomAltPK = '" & BomPK & "'"
        Set rs = New ADODB.Recordset
        rs.Open ssql, ConnUnt, adOpenKeyset, adLockOptimistic
        DefPKroh = rs.Fields("DefPK")

        rs.Close
        ssql = "SELECT DefPK,DefID FROM MMDefinitions WHERE -
        DefPK = '" & DefPKroh & "'"
        Set rs = New ADODB.Recordset
        rs.Open ssql, ConnUnt, adOpenKeyset, adLockOptimistic

        DefIDroh = rs.Fields("DefID")
        rohmat = DefIDroh
        rs.Close
    End If

    Set rs = Nothing

    ConnUnt.Close
    Set ConnUnt = Nothing

End Sub
```

A.2 Simulator Code

```
Dim ssql As String
Dim rs As ADODB.Recordset
Dim strStream As ADODB.Stream
Dim x As Variant
Dim i As Double
Dim ConnUnt As ADODB.Connection

Dim gbes_zu As Boolean, gbes_a As Boolean, gbes_bel As Boolean, -
gbes_palid As String, gbes_lotid As String, -
gbes_fertigmat As String, gbes_rohmat As String, groh_a As Boolean, -
groh_zu As Boolean, groh_bel As Boolean, groh_palid As String, -
groh_lotid As String, groh_fertigmat As String, -
groh_rohmat As String, gum1_a As Boolean, gum2_a As Boolean, -
gfertig_a As Boolean, gfertig_zu As Boolean, gfertig_bel As Boolean, -
gfertig_palid As String, gfertig_lotid As String, -
gfertig_fertigmat As String, gfertig_rohmat As String, -
gbearb_zu As Boolean, gbearb_a As Boolean, gbearb_bel As Boolean, -
gbearb_palid As String, gbearb_lotid As String, -
gbearb_fertigmat As String, gbearb_rohmat As String, -
gmaschine_a As Boolean, gmaschine_bel As Boolean, grob_a As Boolean, -
grob_bel As Boolean


Private Sub Timer1_Timer()

    Set ConnUnt = New ADODB.Connection
    ConnUnt.Provider = "sqloledb.1"
    ConnUnt.Properties("Data Source").Value = "eftpeh"
    ConnUnt.Properties("Initial Catalog").Value = "TestDB"
     ' Datenbank heißt TestDB
    ConnUnt.Properties("Current Language").Value = "German"
```

```
' con_op.SQL_Language, German
ConnUnt.Properties("User ID") = "sa"
' SQL Username ist sa
ConnUnt.Properties("Password") = "mes2007"#
' SQL Passwort ist mes2007
ConnUnt.CursorLocation = adUseClient
ConnUnt.Open

ssql = "SELECT * FROM MES2 "
' hier wird der SQL-Abfrage String auf die Variable ssql
' hinterlegt
Set rs = New ADODB.Recordset
rs.Open ssql, ConnUnt, adOpenKeyset, adLockOptimistic
' das Recordset wird geöfnet und mit den Daten aus der Datenbank
' gefüllt

' Hier werden alle Daten an die globale Variablen übergeben
gbes_zu = rs.Fields("gbes_zu")
gbes_a = rs.Fields("gbes_a")
gbes_bel = rs.Fields("gbes_bel")
gbes_palid = rs.Fields("gbes_palid")
gbes_lotid = rs.Fields("gbes_lotid")
gbes_rohmat = rs.Fields("gbes_rohmat")
gbes_fertigmat = rs.Fields("gbes_fertigmat")
gbes_mengefertig = rs.Fields("gbes_mengefertig")
gbes_mengeroh = rs.Fields("gbes_mengeroh")

gbearb_zu = rs.Fields("gbearb_zu")
gbearb_a = rs.Fields("gbearb_a")
gbearb_bel = rs.Fields("gbearb_bel")
gbearb_palid = rs.Fields("gbearb_palid")
gbearb_lotid = rs.Fields("gbearb_lotid")
gbearb_rohmat = rs.Fields("gbearb_rohmat")
gbearb_fertigmat = rs.Fields("gbearb_fertigmat")
```

```
gbearb_mengefertig = rs.Fields("gbearb_mengefertig")
gbearb_mengeroh = rs.Fields("gbearb_mengeroh")

groh_a = rs.Fields("groh_a")
groh_zu = rs.Fields("groh_zu")
groh_bel = rs.Fields("groh_bel")
groh_palid = rs.Fields("groh_palid")
groh_lotid = rs.Fields("groh_lotid")
groh_rohmat = rs.Fields("groh_rohmat")
groh_fertigmat = rs.Fields("groh_fertigmat")
groh_mengefertig = rs.Fields("groh_mengefertig")
groh_mengeroh = rs.Fields("groh_mengeroh")

gfertig_a = rs.Fields("gfertig_a")
gfertig_zu = rs.Fields("gfertig_zu")
gfertig_bel = rs.Fields("gfertig_bel")
gfertig_palid = rs.Fields("gfertig_palid")
gfertig_lotid = rs.Fields("gfertig_lotid")
gfertig_rohmat = rs.Fields("gfertig_rohmat")
gfertig_fertigmat = rs.Fields("gfertig_fertigmat")
gfertig_mengefertig = rs.Fields("gfertig_mengefertig")
gfertig_mengeroh = rs.Fields("gfertig_mengeroh")

grob_bel = rs.Fields("grob_bel")
grob_a = rs.Fields("grob_a")

gmaschine_bel = rs.Fields("gmaschine_bel")
gmaschine_a = rs.Fields("gmaschine_a")

gum1_a = rs.Fields("gum1_a")
gum2_a = rs.Fields("gum2_a")

rs.Close
Set rs = Nothing
```

```
' das Recordset wird wieder geschlossen und auf Null zurückgesetzt

' Dieser Teil Schliest die Conection wieder und setzt die Variable
' auf Nothing
ConnUnt.Close
Set ConnUnt = Nothing

' Hier werden die Textfelder beschrieben, mit Hilfe der globalen
' Variablen die zuvor ausgelesen wurden
bes_palid.Text = gbes_palid
bearb_palid.Text = gbearb_palid
fertig_palid.Text = gfertig_palid
roh_palid.Text = groh_palid

bes_lotid.Text = gbes_lotid
bearb_lotid.Text = gbearb_lotid
fertig_lotid.Text = gfertig_lotid
roh_lotid.Text = groh_lotid

' Dieser Teil würde nur gebraucht werden wenn auch Rohteile
' bestückt werden müssen
'bes_rohmat.Text = gbes_rohmat
'bearb_rohmat.Text = gbearb_rohmat
'fertig_rohmat.Text = gfertig_rohmat
'roh_rohmat.Text = groh_rohmat

bes_fertigmat.Text = gbes_fertigmat
bearb_fertigmat.Text = gbearb_fertigmat
fertig_fertigmat.Text = gfertig_fertigmat
roh_fertigmat.Text = groh_fertigmat

' Dieser Teil würde nur gebraucht werden wenn auch Rohteile
' bestückt werden müssen
'bes_mengeroh.Text = gbes_mengeroh
```

```
'bearb_mengeroh.Text = gbearb_mengeroh
'fertig_mengeroh.Text = gfertig_mengeroh
'roh_mengeroh.Text = groh_mengeroh

bes_mengefertig.Text = gbes_mengefertig
bearb_mengefertig.Text = gbearb_mengefertig
fertig_mengefertig.Text = gfertig_mengefertig
roh_mengefertig.Text = groh_mengefertig

' Vorerst werden alle MengenTextfelder ausgeblendet
bes_mengeroh.Visible = False
bes_mengefertig.Visible = False
bearb_mengeroh.Visible = False
bearb_mengefertig.Visible = False
fertig_mengeroh.Visible = False
fertig_mengefertig.Visible = False
roh_mengeroh.Visible = False
roh_mengefertig.Visible = False

' Farbe ändern wenn Rohteilpuffer Band aktiv ist
If groh_a = True Then
    roh_a.BackColor = &HFF00&
    roh_palid.BackColor = &HFF00&
    bes_palid.BackColor = &HFF00&
    roh_lotid.BackColor = &HFF00&
    bes_lotid.BackColor = &HFF00&
    roh_rohmat.BackColor = &HFF00&
    bes_rohmat.BackColor = &HFF00&
    roh_fertigmat.BackColor = &HFF00&
    bes_fertigmat.BackColor = &HFF00&
    roh_mengeroh.BackColor = &HFF00&
    bes_mengeroh.BackColor = &HFF00&
    roh_mengefertig.BackColor = &HFF00&
    bes_mengefertig.BackColor = &HFF00&
```

```
Else
    roh_a.BackColor = &H808080
    roh_palid.BackColor = &H808080
    bes_palid.BackColor = &H808080
    roh_lotid.BackColor = &H808080
    bes_lotid.BackColor = &H808080
    roh_rohmat.BackColor = &H808080
    bes_rohmat.BackColor = &H808080
    roh_fertigmat.BackColor = &H808080
    bes_fertigmat.BackColor = &H808080
    roh_mengeroh.BackColor = &H808080
    bes_mengeroh.BackColor = &H808080
    roh_mengefertig.BackColor = &H808080
    bes_mengefertig.BackColor = &H808080

End If

' Farbe ändern wenn Fertigteilpuffer Band aktiv ist
If gfertig_a = True Then

    fertig_a.BackColor = &HFF00&
    fertig_palid.BackColor = &HFF00&
    bearb_palid.BackColor = &HFF00&
    fertig_lotid.BackColor = &HFF00&
    bearb_lotid.BackColor = &HFF00&
    fertig_rohmat.BackColor = &HFF00&
    bearb_rohmat.BackColor = &HFF00&
    fertig_fertigmat.BackColor = &HFF00&
    bearb_fertigmat.BackColor = &HFF00&
    fertig_mengeroh.BackColor = &HFF00&
    bearb_mengeroh.BackColor = &HFF00&
    fertig_mengefertig.BackColor = &HFF00&
    bearb_mengefertig.BackColor = &HFF00&
```

```
Else
    fertig_a.BackColor = &H808080
    fertig_palid.BackColor = &H808080
    bearb_palid.BackColor = &H808080
    fertig_lotid.BackColor = &H808080
    bearb_lotid.BackColor = &H808080
    fertig_rohmat.BackColor = &H808080
    bearb_rohmat.BackColor = &H808080
    fertig_fertigmat.BackColor = &H808080
    bearb_fertigmat.BackColor = &H808080
    fertig_mengeroh.BackColor = &H808080
    bearb_mengeroh.BackColor = &H808080
    fertig_mengefertig.BackColor = &H808080
    bearb_mengefertig.BackColor = &H808080

End If

' Farbe für Umsetzer 1 ändern falls er aktiv ist
If gum1_a = True Then
    um1_a.BackColor = &HFF00&
Else
    um1_a.BackColor = &H808080
End If

' Farbe für Umsetzer 2 ändern falls er aktiv ist
If gum2_a = True Then
    um2_a.BackColor = &HFF00&
Else
    um2_a.BackColor = &H808080
End If

'Zustände oben / unten auf die Test Box der Hubzylinder schreiben
If gbes_zu = True Then
```

```
        bes_zu.Text = "oben"
    Else
        bes_zu.Text = "unten"
    End If

    If gbearb_zu = True Then
        bearb_zu.Text = "oben"
    Else
        bearb_zu.Text = "unten"
    End If

     If gfertig_zu = True Then
        fertig_zu.Text = "oben"
    Else
        fertig_zu.Text = "unten"
    End If

     If groh_zu = True Then
        roh_zu.Text = "oben"
    Else
        roh_zu.Text = "unten"
    End If

    ' Roboter einblenden wenn er aktiv ist
    If grob_a = True Then
        'rob.Picture = -
        'LoadPicture("C:\Diplomarbeit_2007_MES\abb_roboter2.jpg")
        rob.Visible = True

    Else
        rob.Visible = False
    End If

    ' Maschine einblenden wenn sie aktiv ist
```

```
If gmaschine_a = True Then
    'rob.Picture = -
    'LoadPicture("C:\Diplomarbeit_2007_MES\abb_roboter2.jpg")
    maschine.Visible = True

Else
    maschine.Visible = False
End If

' Farbe der Paletten ändern wenn eine vorhanden ist
If bes_palid.Text <> "" Then

    bes_mengeroh.Visible = True
    bes_mengefertig.Visible = True

    bes_palid.BackColor = &H80FFFF
    bes_lotid.BackColor = &H80FFFF
    bes_rohmat.BackColor = &H80FFFF
    bes_fertigmat.BackColor = &H80FFFF
    bes_mengeroh.BackColor = &H80FFFF
    bes_mengefertig.BackColor = &H80FFFF
End If

If bearb_palid.Text <> "" Then

    bearb_mengeroh.Visible = True
    bearb_mengefertig.Visible = True

    bearb_palid.BackColor = &H80FFFF
    bearb_lotid.BackColor = &H80FFFF
    bearb_rohmat.BackColor = &H80FFFF
    bearb_fertigmat.BackColor = &H80FFFF
    bearb_mengeroh.BackColor = &H80FFFF
    bearb_mengefertig.BackColor = &H80FFFF
```

```
    End If

    If fertig_palid.Text <> "" Then

        fertig_mengeroh.Visible = True
        fertig_mengefertig.Visible = True
        fertig_palid.BackColor = &H80FFFF
        fertig_lotid.BackColor = &H80FFFF
        fertig_rohmat.BackColor = &H80FFFF
        fertig_fertigmat.BackColor = &H80FFFF
        fertig_mengeroh.BackColor = &H80FFFF
        fertig_mengefertig.BackColor = &H80FFFF
    End If

    If roh_palid.Text <> "" Then

        roh_mengeroh.Visible = True
        roh_mengefertig.Visible = True
        roh_palid.BackColor = &H80FFFF
        roh_lotid.BackColor = &H80FFFF
        roh_rohmat.BackColor = &H80FFFF
        roh_fertigmat.BackColor = &H80FFFF
        roh_mengeroh.BackColor = &H80FFFF
        roh_mengefertig.BackColor = &H80FFFF
    End If
End Sub
```

Literaturverzeichnis

ANSI/ISA: ANSI/ISA -95.00.01-2000, North Carolina, 2000.

ANSI/ISA: ANSI/ISA -95.00.02-2001, North Carolina, 2001.

ANSI/ISA: ANSI/ISA -95.00.03-2005, North Carolina, 2005.

Kittl, Burkhard: Einsatz von PPS- und Leitsystemen, Wien, 2005.

Kletti, Jürgen: MES – Manufacturing Execution Systems/Moderne Informationstechnologie zur Prozessfähigkeit der Wertschöpfung, Heidelberg, 2006.

Löffelmann, Klaus: Visual Basic.NET – Das Entwicklerbuch, Unterschleißheim, 2004.

Mezzaro, F./Ferrari, R./Pazzini, M.: A&D AS MES SIMATIC IT Production Suite V6 Product Functional Overview, 2005.

Monadjemi, Peter: Visual Basic.NET/Windowsprogrammierung mit dem .NET Framework 1.1 und Visual Studios 2003, München, 2004.

Siemens AG: SIMATIC IT Production Suite V6.1 SP1 Documentation Library, 2005.

VDI: VDI5600/Manufacturing Execution Systems/Fertigungsmanagementsysteme, Düsseldorf, 2006.