

Verhaltenssteuerung in reaktiven Systemen: Konstruktion von Master-Slave-Hierarchien in CIP-Modellen

Hugo Fierz
Institut für Technische Informatik und Kommunikationsnetze
ETH Zürich
Gloriastr.35
CH-8092 Zürich
Tel.: +41 1 632 70 65
Fax: +41 1 632 10 36
E-Mail: fierz@tik.ee.ethz.ch

Hansruedi Müller
CIP System AG
Technoparkstr.1
CH-8005 Zürich
Tel.: +41 1 445 35 90
Fax: +41 1 445 35 91
E-Mail: mueller@ciptool.ch

Abstract: CIP-Modelle beschreiben reaktive Echtzeitsysteme als kooperierende Prozesse, die formal als erweiterte Zustandsmaschinen definiert sind. Mit Master-Slave-Hierarchien können solche Verhaltensmodelle flexibel und problemorientiert strukturiert werden. Das Verhalten eines Prozesses kann durch mehrere alternative Modi beschrieben werden. Jeder Modus wird durch eine Transitionsstruktur von Zustandsübergängen spezifiziert. Eine Master-Slave-Beziehung besteht aus einem Master-Prozess, der durch seinen aktuellen Zustand den aktiven Modus eines Slave-Prozess bestimmt. Ein Moduswechsel des Slave wird damit immer durch einen Zustandsübergang seines Master bewirkt. Mit solchen Verhaltenssteuerungen können hierarchische Verhaltensstrukturen erzeugt werden, die einen wichtigen Teil der Problemstruktur widerspiegeln. Für die grafische Konstruktion von Master-Slave-Hierarchien und deren automatische Umsetzung in ausführbare Software-Komponenten wird das Spezifikationswerkzeug CIP Tool[®] verwendet.

Stichworte: CIP-Methode, modellbasierte Spezifikation, reaktive Systeme, Software-Entwurf, Verhaltenshierarchien

1 Einleitung

Die Aufgabe reaktiver Softwaresysteme ist die Überwachung und Steuerung von Prozessen der Systemumgebung. Derartig eingebettete Echtzeitsysteme bestehen typischerweise aus einer Anzahl kooperierender Programmeinheiten, die ständig mit Aktionen auf auftretende Ereignisse der

Umgebung reagieren müssen [HP85]. Eine Hauptaufgabe bei der Entwicklung solcher Systeme ist eine gut strukturierte Verhaltensbeschreibung zu finden, die einen dynamischen Zusammenhang zum kombinierten Verhalten der externen Prozesse herstellt. Strukturierung durch Verfeinerung ist schwierig anzuwenden, weil elementare Interaktion oft einen direkten Einfluss auf das Verhalten auf höherer Ebene hat. Aufgrund der vielen zyklischen Abhängigkeiten in reaktiven Systemen eignen sich auch Client-Server-Ansätze nur beschränkt, um das Systemverhalten zu strukturieren.

1.1 Hierarchische Strukturen in reaktiven Systemen

In reaktiven Systemen wird oft mit Strukturen gearbeitet, die eine hierarchische Komposition von Systemteilen wie Aktoren, Prozessen oder Funktionsblöcken [SG94, OF94, IEC93] beschreiben. Die verwendete Hierarchierelation beruht auf der bekannten "ist Teil von"-Beziehung, welche von Natur aus azyklisch ist. Solche Kompositionshierarchien werden benützt, um Interaktionsabhängigkeiten auf lokale Kompositionsebenen einzuschränken. Oft führen aber Kompositionshierarchien aufgrund ihrer Starrheit früher oder später zu Strukturkonflikten und "Work Arounds". Solche Schwierigkeiten sind ein Indiz dafür, dass die Verschachtelung von Systemteilen nur teilweise mit der charakteristischen Problemstruktur reaktiver Systeme vereinbar ist.

Eine andere Art von Kompositionshierarchie ist bekannt von hierarchischen Zustandsmaschinenmodellen wie sie Statecharts [H93] oder Modecharts [JL88] anbieten. Die kompositionelle Struktur stellt hier verschachtelte Mengen von Zustandsbereichen dar, die als Superstates bezeichnet werden. Zwischen Superstates können Transitionen definiert werden, die einen Übergang vom einen Zustandsbereich in den andern beschreiben. Im allgemeinen sind Transitionen zwischen Superstates verschiedener Hierarchieebenen möglich. Der Begriff des Superzustate scheint einen Bezug zum in diesem Artikel eingeführten Modus-Begriff zu haben. Der wesentliche Unterschied der beiden Notationen wird im Abschnitt 3.3 diskutiert.

1.2 Master-Slave-Hierarchien in CIP-Modellen

CIP (Communicating Interacting Processes) [FM93, CT95, F199,F199] ist eine formale Entwicklungsmethode für verteilte reaktive Systeme, die sich auf Modelle kooperierender Zustandsmaschinen abstützt. CIP-Modelle werden mit ¹ CIP Tool® [1] konstruiert, indem grafische und textuelle Modellelemente miteinander verknüpft werden. Ein Codegenerator erzeugt aus CIP-Modellen ausführbare Software-Komponenten. Die Grafiken und Modellbeschreibungen in diesem Artikel sind Beispiele automatisch erzeugter Modell-Dokumentation.

Bei der Entwicklung verschiedener industrieller und akademischer Projekte mit CIP ist eine neue hierarchische Struktur entdeckt worden, die als Master-Slave-Hierarchie bezeichnet wird. Das Gesamtverhalten eines CIP-Prozesses kann durch mehrere als Modi bezeichnete alternative Verhalten

¹ Im Tool-Teil dieses Tagungsbandes ist eine kurze Beschreibung von CIP Tool® zu finden. Das Werkzeug wird zudem an der Tool-Ausstellung der Tagung vorgeführt.

spezifiziert werden, Jeder Modus wird durch ein Zustandsübergangsdiagramm beschrieben, welches die Zustände, Input- und Output-Elemente des Prozesses verknüpft. Typische Beispiele solcher Modi sind das Initialisierungs-, Arbeits- und Abschaltverhalten eines Steuerprozesses, die Betriebsarten einer Produktionssteuerung oder die verschiedenen Fehlermodi eines Protokolltreibers.

Das generische Master-Slave-Muster besteht aus einem Master-Prozess, der dynamisch einen Wechsel des aktiven Modus eines Slave-Prozess bewirken kann. Ein Slave-Prozess kann wiederum Master anderer Slave-Prozesse sein. Mit Master-Slave-Beziehungen können, ähnlich wie mit Client-Server-Beziehungen, hierarchische Strukturen konstruiert werden.

Mit Master-Slave-Hierarchien kann nur ein Teil des ganzen Entwurfsproblems gelöst werden. Das Vorgehen ist jedoch gut dazu geeignet, den allgemeinen Entwurfsprozess zu leiten, um zu verständlichen Interaktionsstrukturen reaktiver Systeme zu gelangen.

Im Kapitel 2 geben wir einen kurzen Überblick über die CIP-Notation. Kapitel 3 erklärt, wie in CIP-Modellen Master-Slave-Hierarchien definiert werden. Kapitel 4 illustriert durch ein einfaches aber vollständiges CIP-Modell, wie Master-Slave-Hierarchien funktionieren.

2 CIP Modelle - Eine Einführung

Als Vorbereitung für die Erläuterung von Master-Slave-Hierarchien gibt dieses Kapitel eine Einführung in die Architektur- und Verhaltensmodelle der CIP-Methode.

Ein CIP-Modell besteht aus asynchron kooperierenden Clustern. Jeder Cluster setzt sich aus synchron kooperierenden Prozessen zusammen, die als erweiterte Zustandsmaschinen (Automaten) spezifiziert werden. Das mathematische Modell eines Clusters ist als Produkt dieser Zustandsmaschinen definiert und stellt eine Zustandsmaschine mit mehrdimensionalem Zustandsraum dar, d. h. der Zustand eines Clusters ist durch die Zustände seiner Prozesse gegeben.

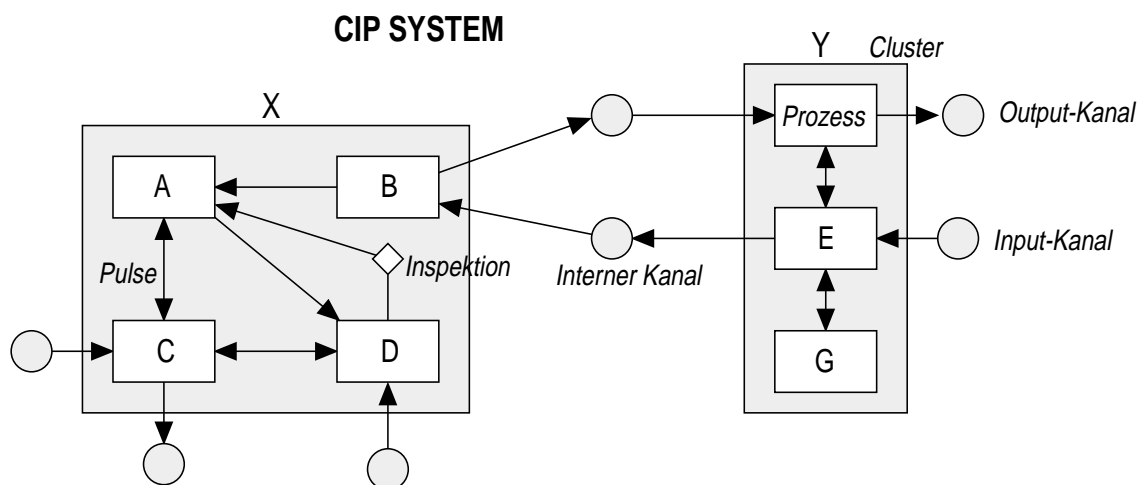


Abbildung 1: CIP-Modell mit zwei Clustern

Die Prozesse der verschiedenen Cluster kommunizieren untereinander und mit der Umgebung über asynchrone Kanäle. Asynchrone Kommunikation bedeutet im CIP-Modell, dass die Schreib- und die Leseaktion einer Übertragung zeitlich getrennt in verschiedenen Clusteraktivierungen ausgeführt werden. Eine Meldung eines Kanals erzeugt einen Zustandsübergang des empfangenden Prozesses. Der so aktivierte Prozess kann durch einen Puls, ein intern erzeugtes Ereignis, weitere Prozesse des Clusters anstossen. Diese können wiederum andere Prozesse durch Pulse aktivieren. Die entstehende verzweigte Interaktionskette ist nicht unterbrechbar und definiert einen einzigen Zustandsübergang des ganzen Clusters. Der Output des Clusters besteht aus Meldungen, die von den aktivierten Prozessen in Kanäle abgesetzt worden sind. Der synchrone Pulsübertragungsmechanismus wird als Interaktion, die asynchrone Übertragung von Meldungen als Kommunikation bezeichnet.

PROCESSES – Erweiterte Zustandsmaschinen

Die Kommunikationsschnittstelle eines Prozesses wird durch einen oder mehrere Inports und Outports definiert. Ein Port wird durch die Menge der zu lesenden oder schreibenden Meldungen spezifiziert. Die Schnittstelle für die synchrone Interaktion innerhalb des Clusters wird durch eine Menge von Impulsen und eine Menge von Outputspulsen gebildet. In Abb. 2 ist die Schnittstelle des Prozesses *Door* beschrieben und sein Verhalten durch eine Transitionsstruktur spezifiziert.

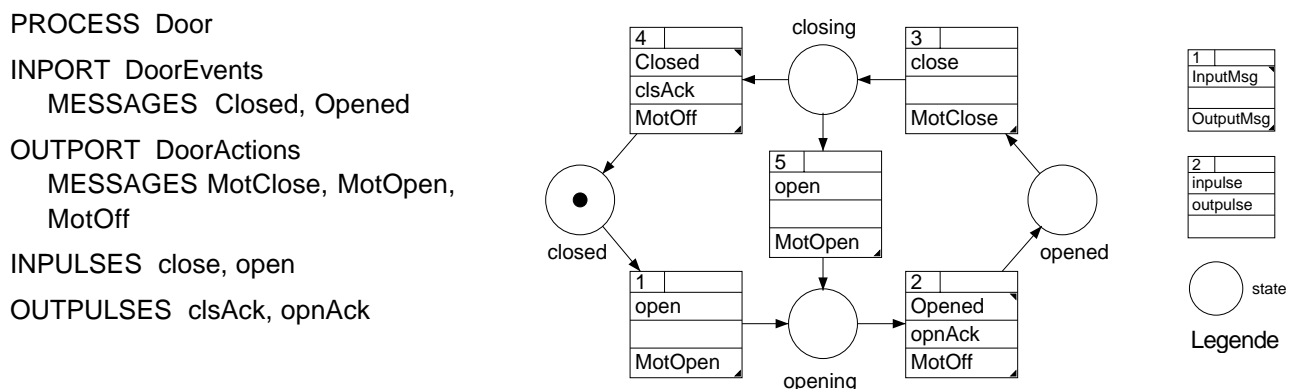


Abbildung 2: Reine Zustandsmaschine

Prozesszustände sind durch Kreise dargestellt, Transitionen durch beschriftete Transitionsboxen. Die Ereignismeldungen *Opened* und *Closed* oder die Impulse *open* und *close* können einen Zustandsübergang auslösen. Jede ankommende Meldung muss erwartet sein, andernfalls wird ein Kontextfehler erzeugt (Exception Handling). Auf Impulse hingegen muss nicht reagiert werden. In jeder Transition kann ein Outputpuls abgesetzt werden, der von mehreren Prozessen des Clusters empfangen werden kann (multi-cast). Zusätzlich kann in jeden Outport eine Meldung geschrieben werden.

Um Daten zu verarbeiten oder Berechnungen auszuführen werden Prozesse als erweiterte Zustandsmaschinen spezifiziert. Die Erweiterung besteht aus statischen Prozessvariablen, Datentypen für Meldungen und Pulse, Operationen und Bedingungen. Operationen werden in aktivierten Transitionen ausgeführt. Bedingungen sind notwendig, damit non-deterministische Verzweigungen eindeutig ausführbar werden. Diese Prozesserweiterungen werden mit bestimmten Konstrukten der

Programmiersprache des generierten Codes definiert und stellen primitive Komponenten eines CIP-Modelles dar.

INTERACTION NET – Synchrone Pulsübertragung

Ein als Interaction Net bezeichneter Graph (Abb. 3) definiert, zwischen welchen Prozessen Pulse übertragen werden. Für jede Prozessverbindung müssen in einer entsprechenden Tabelle gesendete Outputpulse in zu empfangende Inputpulse übersetzt werden (Pulse Translation). Ein Outputpulse kann i. a. von mehreren Prozessen empfangen werden (Multicast).

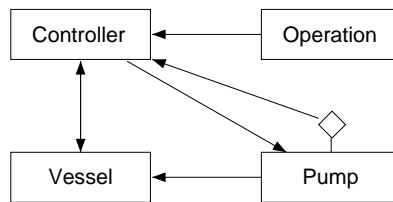


Abbildung 3: INTERACTION NET eines Clusters

Für Modelle mit Multicast-Pulsübertragungen schränkt die definierte Verbindungsstruktur die Interaktionspropagation jedoch zu wenig ein. Weil der Kontrollfluss nicht definiert ist, muss mit Non-Determinismus und zyklischen Kettenreaktionen gerechnet werden (ein bekanntes Problem von Statechart-Modellen). Um eine eindeutige Ausführung von CIP-Modellen sicherzustellen, wird die Pulsübertragung als *sequentieller Multicast* spezifiziert. Die von einem Prozess ausgehenden Interaktionsverbindungen werden als geordnete Menge definiert (Cast Order). Wenn ein Prozess einen Outputpulse absetzt, werden die entsprechenden Empfänger in der spezifizierten Reihenfolge getriggert, was jedesmal eine weiterführende Interaktionskette bewirken kann. Cluster-Transitionen sind damit als eindeutige Prozessaktivierungssequenzen definiert. Das CIP-Werkzeug stellt diese Sequenzen während der Modellkonstruktion automatisch als grafische Interaktionsbäume dar.

Um für Systemreaktionen beschränkte Antwortzeiten garantieren zu können, müssen zyklische Interaktionspfade ausgeschlossen werden. Das Problem wird vom CIP-Werkzeug gelöst, indem nur Modelle mit Interaktionssequenzen zugelassen werden, in denen derselbe Prozess höchstens einmal durch denselben Inputpulse getriggert wird.

STATE INSPECTION – Lesen fremder Prozessvariablen

Bedingungen einer Transitionsstruktur können direkt von den Zuständen und Variablen anderer Prozesse desselben Clusters abhängen. Der Lesezugriff auf die Daten eines anderen Prozesses wird als Zustandsinspektion bezeichnet und erfolgt über Zugriffsfunktionen des inspizierten Prozesses. Durch Zustandsinspektion entsteht eine weitere Abhängigkeit zwischen Prozessen, die im Interaction Net durch Verbindungen mit Rhomben grafisch deklariert sind (Abb. 3). Der Pfeil zeigt zum inspizierenden Prozess (Datenfluss).

COMMUNICATION – Asynchrone Übertragung von Meldungen

Kommunizierenden Prozesse werden im Communication Net über ihre Ports grafisch mit Kanälen verbunden. Input- und Outputkanäle bilden die Schnittstelle zur Umgebung, interne Kanäle sind Teil des CIP-Modells. Kommunizierende Prozesse stellen rezeptive Entitäten dar, die immer bereit sind, Meldungen von Kanälen anzunehmen.

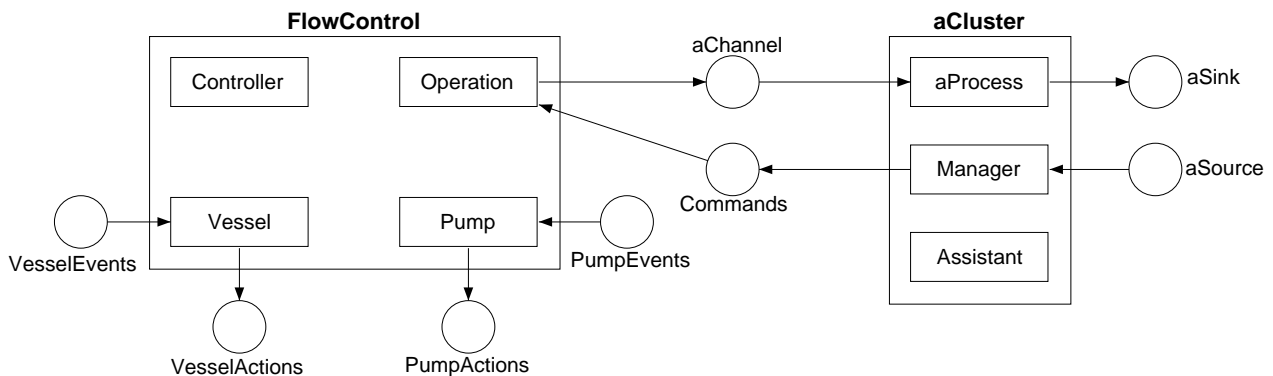


Abbildung 4: COMMUNICATION NET eines CIP Modells

3 Master-Slave-Hierarchien bin CIP Modellen

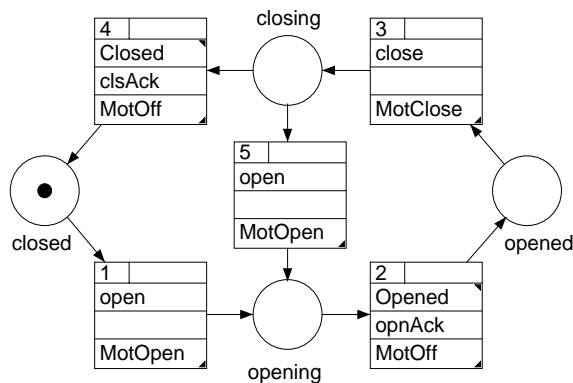
Eine Master-Slave-Beziehung zwischen zwei Prozessen basiert auf alternativen Verhaltensbeschreibungen des Slave-Prozesses. Der Master-Prozess bestimmt dabei durch sein eigenes Verhalten, welches Verhalten vom Slave-Prozess erwünscht ist. Master-Slave-Hierarchien werden durch Konstruktion eines azyklischen Master-Slave-Graphen definiert.

3.1 MODES

Die Transitionsstruktur eines CIP-Prozesses beschreibt, wie der Prozess mit Zustandsübergängen und erzeugten Outputpuls und Meldungen auf Inputmeldungen und Impulse reagiert. Solche Strukturen werden komplex, wenn bestimmte Meldungen oder Impulse einen Einfluss auf das zukünftige Verhalten des Prozesses haben. Ein Alarmmeldung muss zum Beispiel dazu führen, dass sich der Prozess inaktiv bleibt bis der Alarm wieder zurückgesetzt wird. Die resultierende Transitionsstruktur stellt dann eine Superposition von zwei Strukturen dar, eine für das Normal- und eine für das Alarmverhalten. Um solche strukturelle Verflechtungen zu verhindern, wird das Verhalten des Prozesses durch alternative Modi spezifiziert. Jeder Modus wird grafisch durch ein Transitionsdiagramm beschrieben, welches die Zustände, Meldungen und Pulse des Prozesses verknüpft.

Abb. 5 zeigt eine Erweiterung des *Door* Prozesses von Abb. 2 mit einen zusätzlichen *shutting* Modus. Dieser Modus beschreibt ein zum Normalfall alternatives Verhalten, bei dem möglichst schnell die Türe geschlossen und nicht mehr geöffnet wird.

MODE normal



MODE shutting

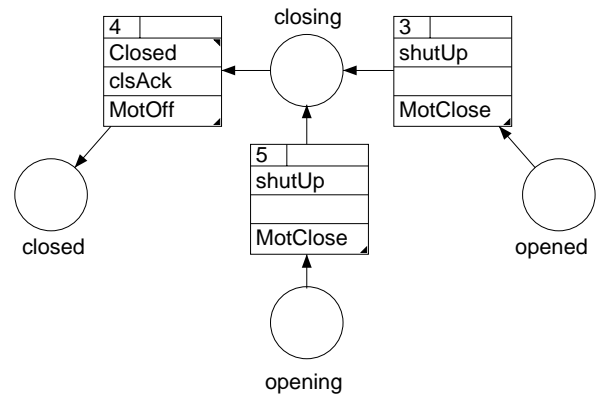


Abbildung 5: Two modes of process Door

Ein Moduswechsel eines Prozesses kann durch einen oder mehrere als Master spezifizierte Prozesse desselben Clusters bewirkt werden. Der beeinflusste Prozess wird als Slave bezeichnet. Die Semantik einer Master-Slave-Beziehung ist wie folgt definiert:

Die aktuellen Zustände der Master-Prozesse bestimmen den aktiven Modus des Slave-Prozesses.

Ein Moduswechsel eines Slave kann immer dann auftreten, wenn ein Master seinen Zustand wechselt. Selbst wenn sich der *Door* Prozess von Abb. 5 im *normal* Modus im Zustand *closed* befindet, kann ein Master ein Umschalten in den *shutting* Modus bewirken. Der Effekt dieser Verhaltensänderung ist, dass die Türe auch dann geschlossen bleibt, wenn dem Prozess ein *open* Puls gesendet wird,

Ein Slave kann indirekt bei sich selbst einen Moduswechsel bewirken, indem er bei einem seiner Master mit einem Puls einen Zustandsübergang auslöst. Dieses Verhaltensmuster tritt typisch bei Fehlerverarbeitungen auf: Wenn ein Slave einen Fehler erkennt, informiert er mit einem Fehler-Puls seinen Error-Master, der einen Wechsel in einen adäquaten Fehlermodus bewirken kann.

Ein Moduswechsel verändert den aktuellen Zustand des Slave-Prozesses nicht. Ein Zustandsübergang kann nur durch eine Transition im aktiven Modus erzeugt werden. Der Zusammenhang zwischen dem aktuellen Zustand und der Interaktions- und Kommunikationsgeschichte des Prozesses bleibt damit erhalten. Dies wird besonders wichtig, wenn der aktuelle Prozesszustand ein Abbild eines Zustandes der Umgebung ist.

3.2 MASTER-SLAVE-Beziehungen

Die Master-Slave-Beziehungen eines Clusters werden wie in Abb. 6 in einem entsprechenden Master-Slave-Graphen definiert. Master-Slave Beziehungen sind durch Dreieck-Symbole dargestellt, die oben mit Master-Prozessen und unten mit dem Slave-Prozess verbunden sind. Der gerichtete Graph ist immer azyklisch, damit die allgemeine Hierarchiebedingung erfüllt ist. Die im Hierarchiegraphen (Abb. 6) angedeuteten Hierarchieebenen haben keine formale Bedeutung. Solche Ebenen

werden oft in informaler Weise dazu benutzt, um Prozesse zu gruppieren, die auf einer gemeinsamen Abstraktionsebene interagieren. Normalerweise sind nicht alle Prozesse eines Cluster in die Master-Slave-Hierarchie eingebunden.

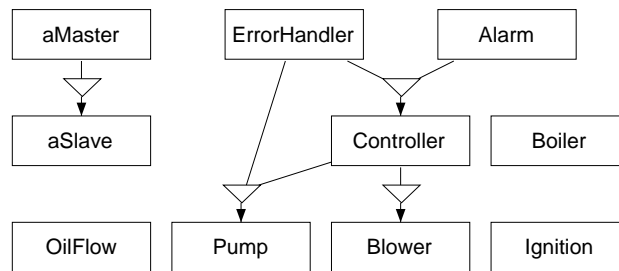


Abbildung 6: MASTER-SLAVE Hierarchie Graph

Der Zusammenhang zwischen Master-Zuständen und Slave-Modi wird mit sog. *MODE SETTING*-Tabellen spezifiziert. Ein solche Tabelle definiert eine funktionale Abbildung vom kartesischen Produkt der Master-Zustandsmengen auf die Menge der Modi des entsprechenden Slave.

Die Master-Slave-Semantik würde auch zyklische Master-Slave-Relationen zulassen. Der Sinn der azyklischen Einschränkung wird klar, wenn Master-Slave-Beziehungen aus einer problemorientierten Sicht betrachtet werden: Ein Master, der das Verhalten eines Slave steuert, wird zu einer höheren Verhaltensstufe gehören als sein Slave. Zyklische Master-Slave-Abhängigkeiten würden damit schnell zu einem schwer verständlichem Systemverhalten führen,

Der Zweck einer "guten" hierarchischen Struktur kann immer durch den Aspekt der zugrundeliegenden Hierarchierelation charakterisiert werden. Kompositionshierarchien basieren auf einer "ist Teil von"-Relation, der Aspekt ist Gruppieren und Verfeinern. Die klassische Programm-Hierarchie von Parnas [P74] beruht auf einer "wird benutzt"-Relation, welche einen Dienstaspekt ausdrückt. Dasselbe gilt für Client-Server Hierarchien von objektorientierten Modellen. Die Aspekt von Master-Slave-Relationen wird durch "One man's state is another man's process" ausgedrückt. Der Zweck von Master-Slave-Hierarchien ist abstrakte Verhaltenssteuerung. Das Umschalten eines Modus ändert das reaktive Verhalten eines Prozesses während elementare Interaktionen lediglich zu Zustandsänderungen im aktiven Modus führen.

3.3 Vergleich mit Statecharts und Modecharts

Der vorgestellte Modus-Begriff unterscheidet sich wesentlich von entsprechenden Begriffen in hierarchischen Zustandsmaschinenmodellen, wie z. B. *Or-Superstates* in Statecharts [H93] oder *Serial Modes* in Modechart [JL88]. Superstate-Transitionen in Statecharts beschreiben ebenfalls einem Verhaltenswechsel. Aber im Unterschied zu den CIP-Modi ist das Verhalten in Superstates auf disjunkten Bereichen der darunterliegenden Zustandsebene definiert. Eine Superstate-Transition führt damit immer auch zu einer Änderung aller Zustände der unteren Ebenen. Solche Modelle machen es schwierig, Verhaltensänderung zu modellieren, bei der die gespeicherte Kontextinformation,

repräsentiert durch aktuelle Zustände, erhalten bleiben soll. Der Unterschied kann gut am Beispiel einer spielenden Fussballmannschaft erläutert werden, bei welcher der Trainer während des Spiels eine Änderung des taktischen Verhalten bewirken will. Die Superstate Transition: Um die Taktik zu ändern muss das Spiel unterbrochen und die Mannschaft neu aufgestellt werden. Der Master-Slave-Mechanismus: Der Trainer gibt ein bestimmtes Zeichen, damit die spielende Mannschaft das taktische Verhalten ändert.

4 CIP Fallstudie: LiquidServer

Das dargestellte ausführbare CIP-Modell wurde mit CIP Tool® spezifiziert. Die Modellbeschreibung entspricht einem automatisch erzeugten Modell-Report.

Anlage

Mit dem LiquidServer-System sollen einem verfahrenstechnischen Prozess automatisch portionierte Mengen einer Flüssigkeit zugeführt werden können. Ein Messkessel ist mit zwei diskreten Niveau-Sensoren ausgestattet, welche die Füllzustände voll und leer detektieren. Ein Ablass-Ventil kann durch das System geöffnet und geschlossen werden. Mit einer steuerbaren Pumpe wird der Kessel gefüllt; das Zuflussventil der aktivierten Pumpe darf erst geöffnet werden, wenn der Solldruck erreicht worden ist.

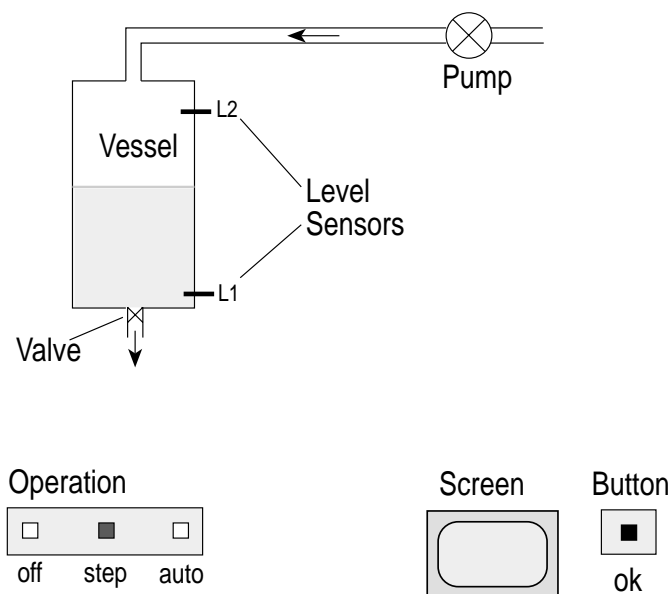


Abbildung 7: Anlage LiquidServer

Funktionale Anforderungen

Kesselfüllungen werden in zwei verschiedenen Betriebsarten zugeführt: In der interaktiven Step-Betriebsart bestätigt ein Bediener für jede einzelne Kesselfüllung eine Bildschirm-Anfrage. In der Auto-Betriebsart werden ständig Kesselfüllungen abgelassen.

Der gedrückte Knopf eines 3-Knopf-Schalters bestimmt die aktuelle Betriebsart. Das System muss in der Step-Betriebsart gestartet werden. Der Bediener kann danach zu jedem Zeitpunkt die Betriebsart wechseln. Wenn das System ausgeschaltet wird, muss die aktuelle Kesselfüllung noch vollständig zugeführt werden.

4.1 CIP-Modell

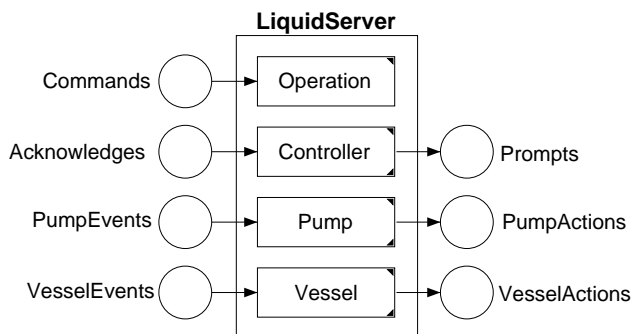
Bemerkung: Alle Prozesse des Systems haben höchstens einen Inport und einen Outport; der Einfachheit halber werden deshalb im CIP-Modell keine Port-Namen verwendet.

Legende: Die Eckmarkierungen bei Prozessboxen zeigen an, ob der Prozess externen Input bzw. Output hat. Obere Ecke: Kanalinput; Untere Ecke: Kanaloutput. Analoge Markierung treten bei den Transitionsboxen auf. Die Anfangszustände der Prozess sind mit einem Punkt markiert.

SYSTEM LiquidServerSystem

Die CIP-Methode wendet die Modellierungskonzepte der JSD-Methode (Jackson System Development) [C89] an. Als Erstes werden sog. Modellprozesse definiert, welche mit den externen Prozessen der Umgebung kommunizieren. Modellprozesse heissen sie darum, weil ihre Transitionsstrukturen das Verhalten der externen Prozesse modellieren. Im Weiteren werden dann Funktionsprozesse eingeführt, die mit den Modellprozessen interagieren und kommunizieren, um so das angeforderte Systemverhalten zu erzeugen. Das CIP-Modell des Fallbeispiels hat nur einen einzigen Cluster, bestehend aus vier Prozessen: *Pump*, *Vessel* und *Operation* sind Modellprozesse, deren Verhalten durch den Funktionsprozess *Controller* koordiniert wird.

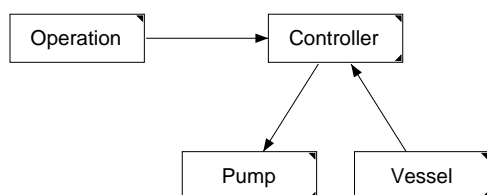
COMMUNICATION NET



Das Kommunikationsnetz enthält lediglich Input- und Outputkanäle, welche die Schnittstelle zur Umgebung modellieren.

CLUSTER LiquidServer

INTERACTION NET



Das Interaktionsnetz definiert Verbindungen für Pulsübertragungen. Der Prozess *Operation* muss den *Controller* triggern, wenn das System gestartet wird. Der *Controller* ist zuständig für das Füllen und Leeren und muss vom Prozess *Vessel* über Änderungen des Füllzustandes informiert werden.

PULSE TRANSLATIONS

SENDER Controller

pumpOff -> Pump.pumpOff

pumpOn -> Pump.pumpOn

SENDER Operation

start -> Controller.start

SENDER Vessel

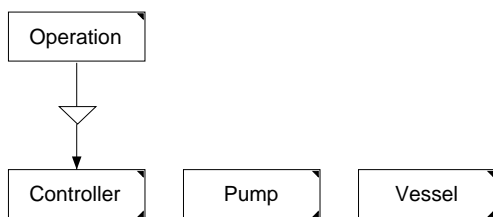
emptied -> Controller.emptied

filled -> Controller.filled

Pulse Translations verknüpfen Outpulse eines Senders mit Impulsen eines Empfängers. Verknüpfte Outpulse und Impulse sind in diesem Beispiel gleich benannt worden.

Weil in diesem Beispiel kein Multicast auftritt, wird keine spezifische Cast-Ordnung definiert.

MASTER-SLAVE GRAPH



Das Verhalten des *Controller* ist den Betriebsarten entsprechend durch verschiedene Modi beschrieben, Der Prozess *Operation* bestimmt die aktuelle Betriebsart und tritt als Master des *Controller* auf.

MODE SETTING Controller

	MASTER Operation		
STATES	idle	stepping	automatic
MODES	shutdown	interactive	continued

Für jede Master-Slave-Beziehung ist in einer Tabelle der funktionale Zusammeng zwischen Master-Zuständen und Slave-Modi spezifiziert.

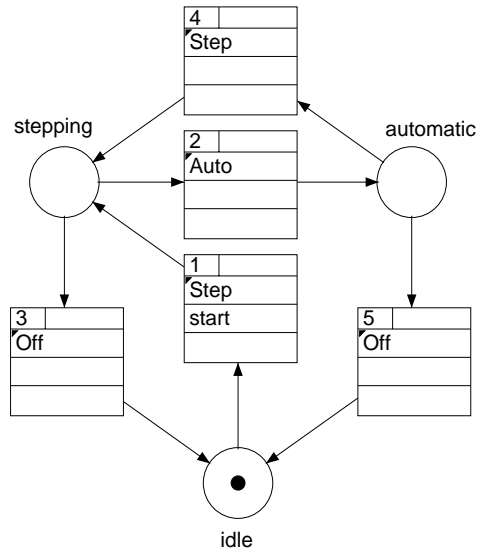
PROCESS Operation

IMPORT MESSAGES Auto, Off, Step

OUTPUTS start

Der Prozess *Operation* modelliert den 3-Knopf-Schalter. Ein Zustandsübergang bewirkt immer ein Moduswechsel des *Controller* (i. a. muss nicht jeder Zustandsübergang einen Moduswechsel bewirken). Mit der Transition 1 startet der Prozess den *Controller*, der den *start*-Puls bereits im Modus *interactive* empfängt.

MODE normal



PROCESS Controller

IMPORT MESSAGES Ok

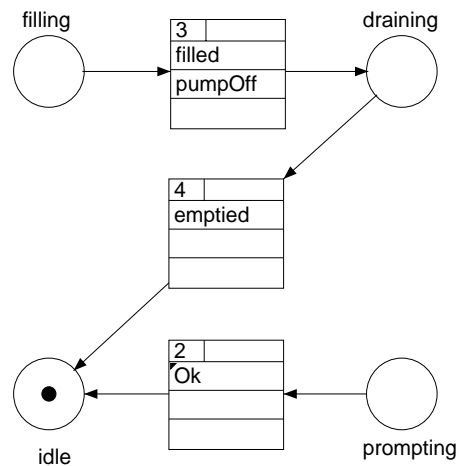
OUTPUT MESSAGES Prompt

INPUTS emptied, filled, start

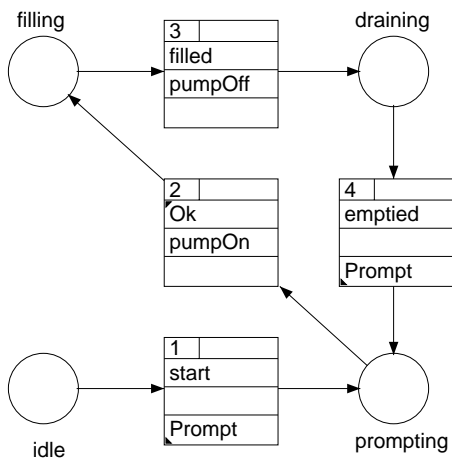
OUTPUTS pumpOff, pumpOn

Die drei Modi beschreiben, wie in den beiden Betriebsarten oder beim Abschalten das System gesteuert wird.

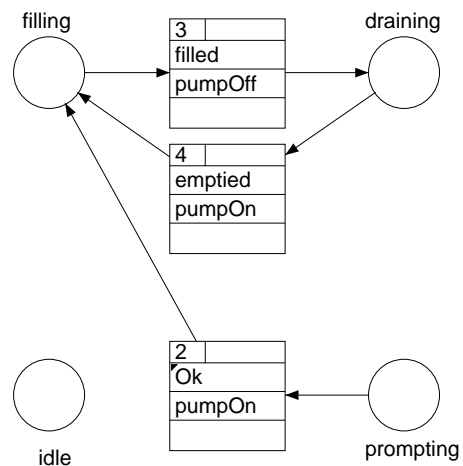
MODE shutdown



MODE interactive



MODE continued



PROCESS Pump

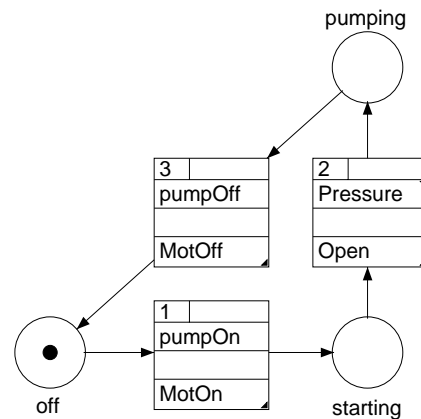
INPORT MESSAGES Pressure

OUTPORT MESSAGES MotOff, MotOn, Open

INPULSES pumpOff, pumpOn

Der Prozess startet und stoppt die Pumpe. Die Ereignismeldung *Pressure* tritt auf, wenn der Solldruck erreicht worden ist.

MODE enabled



PROCESS Vessel

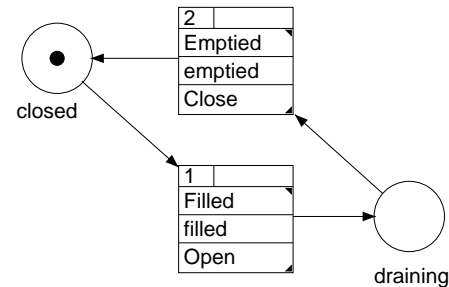
INPORT MESSAGES Emptied, Filled

OUTPORT MESSAGES Close, Open

OUTPULSES emptied, filled

Der Prozess modelliert sehr einfach das Verhalten des Kessels und meldet jede Zustandsänderung dem *Controller*.

MODE working



4.2 Animation des LiquidServer-Modells

Das System kann animiert werden, indem Input-Meldungen "eingegeben" werden. Die Zustandsübergänge der Prozesse können visualisiert werden, indem Spielsteine in den Transitionsstrukturen verschoben werden. Mit CIP Tool® können automatisch erzeugte Animationen direkt am Bildschirm verfolgt werden.

Als Beispiel beschreiben wir die durch die Input-Sequenz *Step, Ok, Pressure, Auto, Filled* erzeugte Animationsgeschichte (Trace). Jede Tabelle beschreibt eine Clustertransition.

PROCESS	MODE	PRESTATE	INPUT	POSTSTATE	OUTPULSE	OUTMSG
Operation	normal	idle	Step	stepping	start	-
Controller	interactive	idle	start	prompting	-	Prompt

Controller	interactive	prompting	Ok	filling	pumpOn	-
Pump	enabled	off	pumpOn	starting	-	MotOn

PROCESS	MODE	PRESTATE	INPUT	POSTSTATE	OUTPUTPULSE	OUTMSG
Pump	normal	starting	Pressure	pumping	-	-
Operation	normal	stepping	Auto	automatic	-	-
Vessel	working	draining	Filled	draining	filled	Open
Controller	continued	filling	filled	draining	pumpOff	-
Pump	enabled	pumping	pumpOff	off	-	MotOff

Tabellen: Fünf Clustertransitionen einer Animation

4.3 System-Erweiterung

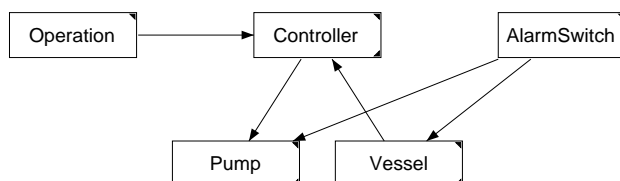
Um zu illustrieren, wie komplexe Master-Slaver-Hierarchien entstehen wird das LiquidServer-System mit einem Alarmverhalten erweitert. Das System wird dazu mit einem weiteren 3-Knopf-Schalter (Normal, Alarm, Rescue) ausgerüstet. Wenn der Normal-Knopf gedrückt ist, arbeitet das System wie in der nicht erweiterten Version. Mit dem Alarm-Knopf soll das arbeitende System sofort unterbrochen werden können. Die aktivierte Pumpe darf aber erst ausgeschaltet werden, wenn der Solldruck erreicht worden ist. Um wieder zum Normalverhalten zu gelangen muss zuerst der Rescue-Knopf gedrückt werden. Das System muss dann in der Step-Betriebsart arbeiten bis der Normal-Knopf gedrückt wird.

Vom erweiterten CIP-Cluster werden nur die veränderten grafischen Modelle und die neuen MODE SETTING-Tabellen dargestellt.

CLUSTER LiquidServerWithAlarm

Für den Alarmschalter kommt neu der Prozess *AlarmSwitch* dazu. Die Interaktionsstruktur muss nur bezüglich dieses Prozesses erweitert werden. Das Verhalten des *Controller* ist von der Erweiterung nicht betroffen. Die Prozesse *Pump* und *Vessel* sind je mit einem zusätzlichen Modus für den Alarmfall versehen worden.

INTERACTION NET



PULSE TRANSLATIONS

SENDER Controller
 pumpOff -> Pump.pumpOff
 pumpOn -> Pump.pumpOn

SENDER Operation
 start -> Controller.start

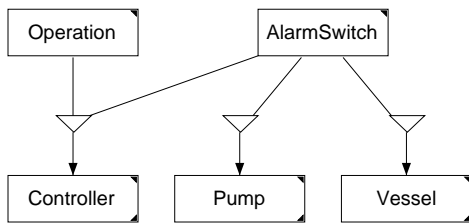
SENDER Vessel
 emptied -> Controller.emptied
 filled -> Controller.filled

SENDER AlarmSwitch
 go -> Pump.go, Vessel.go
 stop -> Pump.stop, Vessel.stop

CAST ORDER

SENDER AlarmSwitch
 RECEIVERS Pump, Vessel

MASTER-SLAVE GRAPH



Das Verhalten des Controller ist zusätzlich abhängig vom Modellprozess *AlarmSwitch*, weil der gedrückte Rescue-Knopf nur ein Arbeiten im *Step* Modus zulässt. Auch das Verhalten der Modellprozesse *Vessel* und *Pump* hängt vom Prozess *AlarmSwitch* ab, da die Aktivität der entsprechenden Geräte bei einem Alarm unterbrochen werden muss.

MODE SETTING Controller

	MASTER AlarmSwitch		
STATES	normal	rescue	alarm
idle	shutdown	shutdown	shutdown
stepping	interactive	interactive	interactive
automatic	continued	interactive	interactive
MASTER Operation			

MODE SETTING Pump

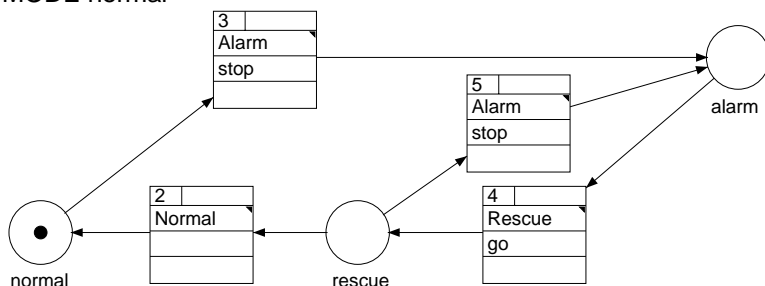
	MASTER AlarmSwitch		
STATES	normal	rescue	alarm
MODES	enabled	enabled	disabled

MODE SETTING Vessel

	MASTER AlarmSwitch		
STATES	normal	rescue	alarm
MODES	working	working	blocked

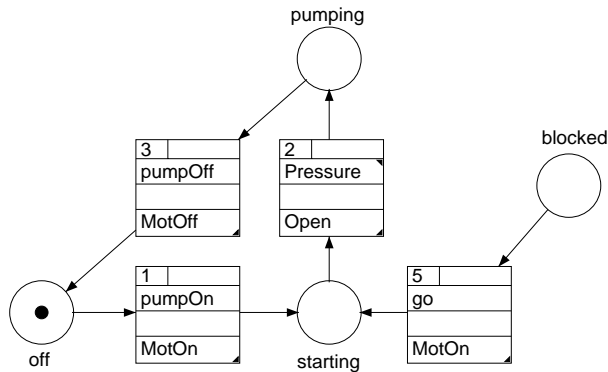
PROCESS AlarmSwitch

MODE normal

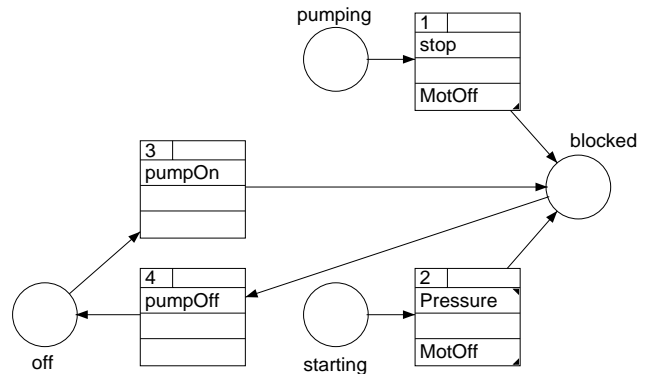


PROCESS Pump

MODE enabled



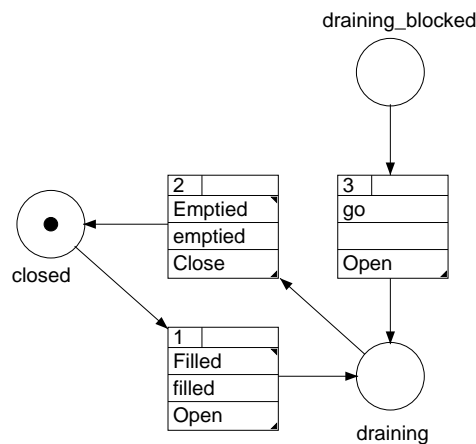
MODE disabled



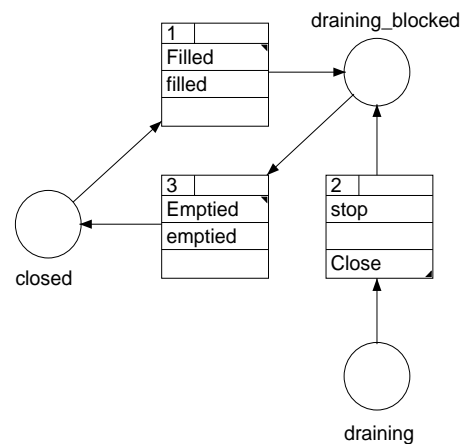
Der Prozess steuert aufgrund von Pulsen des *Controller* die Pumpe. Selbst im *disabled* Modus müssen aufgrund von externen Race-Conditions (gleichzeitige Ereignisse) und der physikalischen Trägheit der externen Prozesse Pulse vom *Controller* erwartet werden.

PROCESS Vessel

MODE working



MODE blocked



Das Leeren des Kessels wird unterbrochen, wenn ein Alarm auftritt. Auch hier müssen im *blocked* Modus aufgrund von Race-Conditions und der physikalischen Trägheit Kessel-Ereignisse erwartet werden.

5 Schlusswort

Master-Slave-Hierarchien sind ein Strukturierungsmittel, das auf Verhaltensabstraktion beruht. Das Verhalten eines Slave-Prozesses wird durch verschiedene alternative Modi definiert. Jeder Modus beschreibt durch eine Transitionsstruktur, wie aufgrund von elementaren Interaktionen der Prozesszustand ändert und neue Interaktionen erzeugt werden. Der aktive Modus wechselt, wenn ein zugeordneter Master-Prozess bestimmte Zustandsübergänge ausführt. Dieser Einfluss des Master auf den Slave wird als Interaktion höherer Ordnung bezeichnet, weil nicht einfach ein Zustandsübergang

des Slave, sondern einen Wechsel des Prozessverhaltens bewirkt wird. Ein induzierter Moduswechsel stellt eine abstraktere Beeinflussung dar als ein durch Interaktion erzeugter Zustandsübergang. Das Master-Slave-Konzept bekommt damit eine Bedeutung als starkes Strukturierungsmittel.

Die Verwendung von Master-Slave-Hierarchien hilft mit Verhaltensbeschreibungen reaktiver Systeme übersichtlicher zu gestalten. Master-Slave-Beziehungen beschreiben Verhaltensabhängigkeiten, die einen wichtigen Teil der funktionalen Struktur widerspiegeln. Verhaltensbeschreibungen mittels alternativer Modi entsprechen einem weiteren Teil der Problemstellung. Informale Beschreibungen von Modi treten meistens bereits in der funktionalen Anforderungsspezifikation auf.

Eine Hauptaufgabe eines reaktiven Systems ist, ständig einen aktuellen Bezug zum externen Kontext zu unterhalten. Die separierte Beschreibung von Interaktionen höherer Ordnung erlaubt die Dynamik der Kontextabhängigkeiten verständlich in den Griff zu bekommen. Bei einer Modusänderung bleibt der Zustand des Prozesses und damit der Bezug zum seinem Kontext erhalten, während der Zweck elementarer Interaktionen das Nachführen des sich ändernden Kontextbezuges ist. Diese Unterscheidung wird besonders wichtig, wenn "Race Conditions" aufgrund gleichzeitig auftretende Ereignisse behandelt werden müssen.

Der statische Zusammenhang zur Problemstruktur hilft zudem beim Systemunterhalt und bei der Wiederverwendung von Systemteilen. Anstehende Änderungen können in der hierarchischen Struktur von Master-Slave-Modellen wesentlich einfacher eingebracht werden als in Systemen, bei denen alle Abhängigkeiten über elementare Interaktionen erzeugt werden. Ein Slave mit mehreren Modi kann zudem leicht mit einem anderen Master wiederverwendet werden. Neue Modi können ohne Seiteneffekte spezifiziert werden. Aber auch die Wiederverwendung von kooperierenden Prozessgruppen bietet kein grosses Problem, wenn die zugehörige Master-Slave-Struktur im neuen System integriert werden kann.

Master-Slave-Hierarchien sind in verschiedenen industriellen und akademischen Projekten angewendet worden. Oft wurden zusätzlich informale Hierarchieebenen festgelegt. Bei der Entwicklung eines Hybrid-Autos [T97] zum Beispiel haben die folgenden Hierarchieebenen geholfen, die komplexe Interaktionsstruktur des Systems in den Griff zu bekommen:

- Ebene 5: Fehlerbehandlung
- Ebene 4: Leitfunktionen
- Ebene 3: Umschalten des Energie Flusses
- Ebene 2: Regulierung des Energie Flusses
- Ebene 1: Steuerung und Regelung der Energieproduzenten

Unsere aktuelle Forschungsarbeit sucht nach generischen Zusammenhängen zwischen elementaren Interaktions- und Master-Slave-Strukturen. Ob geeignete Regeln gefunden werden können, mit welchen die elementaren Interaktionen durch die Master-Slave-Struktur eingeschränkt werden kann ist eine offene Frage. Wir erwarten, dass hier dem Begriff der Hierarchieebenen eine zentrale Rolle zukommen wird.

Literatur

- [C89] CAMERON, J.R.: *The modelling phase of JSD', JSP and JSD: The Jackson Approach to Software Development*. (IEEE Computer Society Press, 1989), pp. 282-292
- [CT95] *User Manual CIP Tool®*. CIP System AG, CH-8005 Zürich, 1995-99
Internet: <http://www.ciptool.ch>
- [FM93] FIERZ, H., MÜLLER, H. and NETOS, S.: *CIP - Communicating Interacting Processes*. Proceedings SAFECOMP'93, 1993, Poznan-Kiekrz Poland, Springer Verlag, pp. 361-370
- [F199] FIERZ, H.: *Die CIP-Methode: Modellbasierte Entwicklung eingebetteter Echtzeitkomponenten..* Tagungband Embedded Intelligenc '99. Kongress der Embedded Systems. Messezentrum Nürnberg, 1999.
- [F299] FIERZ, H.: *The CIP Method: Component- and Model-Based Construction of Embedded Systems*. Submitted to European Software Engineering Conference 1999 – ESEC 99.
- [HP85] HAREL, D. and PNUELI, A.: *On the Development of Reactive Systems*. Logics and Models of Concurrent Systems, Springer New York, 1985, pp. 477-499
- [H93] HAREL, D.: *Statecharts: A Visual Formalism for Complex Systems*. Science of Computer Programming, 1987, 8(3) pp. 231-274
- [SG94] SELIC, B., GULLEKSON, G. and WARD, P.T.: *Real-Time Object-Oriented Modeling*. (John Wiley & Sons, New York 1994)
- [IEC93] 'Programmable controllers - Programming languages', International Standard IEC 1131-3, (International Electrotechnical Commission, 1993)
- [JL88] JAHANIAN, F., LEE, R.S. and MOK, A. : *Semantics of Modechart in Real-Time Logic*. Proc. 21st Hawaii International Conference on Systems Science, 1988, 2, pp. 479-489
- [OF94] OLSEN, A., FAERGEMAND, O., MOELLER, B., REED, R. and SMITH, J.R.W.: *Systems Engineering Using SDL-92*. (Elsevier, Amsterdam 1994)
- [P74] PARNAS, D.: *On a 'Buzzword': Hierarchical Structure*. Information Processing 1974, North-Holland, Amsterdam, pp. 336-339
- [T97] TRUTMANN, HO.: *Flexible Small Scale Parallel System in the Development of a Hybrid Vehicle*. Proc. Parallel and Distributed Processing Techniques and Applications, CSREA, Las Vegas, Nevada, 1997, 2, pp. 684-6911.