

# Die CIP-Methode

## Modellbasierte Entwicklung eingebetteter Echtzeitkomponenten

Hugo Fierz, Institut TIK, ETH Zürich  
Hansruedi Müller, CIP System AG, Zürich

CIP (Communicating Interacting Processes) ist eine formale Entwicklungsmethode, mit der eingebettete Systeme durch grafische Modelle ablauffähig spezifiziert werden können. Der Entwicklungsansatz der Methode geht vom Verhalten der realen Prozesse der Umgebung aus und führt in konstruktiven Schritten zum kompositionell definierten Gesamtsystem. Das auf die Methode zugeschnittene Spezifikationswerkzeug *CIP Tool*<sup>®</sup> mit grafischen Editoren und Code-Generatoren erlaubt eine effiziente Anwendung der Methode in der Praxis.

### 1. Einleitung

Dieser Artikel gibt eine Einführung in die CIP-Methode [2]. Mit CIP wird das reaktive Verhalten eingebetteter Systeme durch formale grafische Modelle spezifiziert. Eingebettete Systeme sind Rechnersysteme, die als integrierender Bestandteil von Geräten und Anlagen für die Steuerung der umgebenden Prozesse verantwortlich sind. In der Regel haben derartige Systeme speziellen Ansprüchen hinsichtlich Zuverlässigkeit und Einhaltung von Echtzeitbedingungen zu genügen. Das notwendige ingenieurmässige Arbeiten wird durch teilformale Modellierungstechniken (Strukturierte Analyse, UML) jedoch zu wenig unterstützt. Das Verhalten eines Systems kann mit solchen Methoden erst in der Implementationsphase durch die kompilierbaren Programme vollständig definiert werden. Die Folge sind oft Produkte, welche die hohen Qualitätsanforderungen nicht zufriedenstellend erfüllen, was sich schliesslich in unnötig hohen Unterhaltskosten niederschlägt. Mit CIP wird eine entscheidende Verbesserung der Software-Qualität möglich, weil bereits in den problemorientierten Entwicklungsphasen mit formalen Modellen gearbeitet wird. Weil die Problemlösungen im Kontext der Anforderungen gefunden werden, entstehen naturgemäss weniger Entwurfsfehler. Bekanntlich kommen solche Fehler besonders

teuer zu stehen, wenn sie erst am implementierten System entdeckt werden. Zudem erlaubt die formale Arbeitsweise den Entwicklungsprozess konsequenter durch Softwarewerkzeuge zu unterstützen.

In Kapitel 2 wird kurz das Entwicklungsproblem eingebetteter Systeme charakterisiert und eine generische Problemzerlegung beschrieben. Kapitel 3 erklärt die Modellierungskonstrukte der CIP-Methode. Kapitel 4 geht auf den Entwicklungsprozess von CIP ein und Kapitel 5 beschreibt die Implementation von CIP-Modellen. Ein illustratives Beispiel zeigt in Kapitel 6, wie die Methode angewendet wird. Das Papier schliesst mit einer kurzen Zusammenfassung der beschriebenen Konzepte.

### 2. Generische Problemzerlegung

Die Umgebung eines eingebetteten Systems besteht typischerweise aus mehreren realen Prozessen, jeder mit einem eigenen aktiven Verhalten. Bei physikalischen Prozessen ist dieses Verhalten in erster Linie durch die physikalische Dynamik bestimmt. Bei Prozessen, die bereits durch lokale Mikroprozessoren gesteuert sind, ist das Verhalten im Wesentlichen durch die integrierte Software gegeben. Damit ein

System auch durch den Menschen beeinflusst werden kann, muss die Anlage zusätzlich mit entsprechenden Bedienungsgeräten ausgestattet sein.

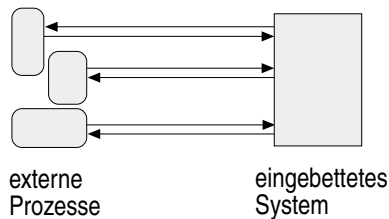


Bild 1 – Eingebettetes System mit Umgebung

## 2.1 Problemzerlegung

Bei der Entwicklung eingebetteter Systeme müssen grundsätzlich zwei Aufgaben gelöst werden:

- Korrekte Reaktion auf Ereignisse der Umgebung
- Erfassen des Prozessverhaltens und Erzeugen der Einwirkungen

Die erste Aufgabe wird im Wesentlichen durch die funktionalen Anforderungen definiert, welche das erwünschte Verhalten des Gesamtsystems beschreiben. Solche Anforderungen beziehen sich immer auf Prozessphänomene wie Ereignisse, Aktionen und Zustände, die hier als *essentielle Phänomene* bezeichnet werden. Verhaltensanforderungen stellen meistens kausale oder zeitliche Bedingungen zwischen solchen Erscheinungen dar.

Die zweite Aufgabe betrifft das Erfassen und Bewirken der essentiellen Phänomene mittels Sensoren und Aktoren. Die Informationsübertragung zwischen Sensoren/Aktoren und Rechner ist stark technologieabhängig. Typisch sind Koppelungen über gemeinsame Variablen oder Feldbusmeldungen.

Es liegt in der Natur der Sache, dass man versucht, das funktionale Problem unabhängig von der Sensor/Aktor-Technik zu lösen. Das wird sogar notwendig, wenn das Systemverhalten in verschiedenen Test-Umgebungen wie Simulationsmodellen, Prüfständen und Teilen des Zielsystems validiert werden muss. Durch eine frühe Stabilisierung der Schnittstellen zwischen funktionalem und Verbindungsproblem wird es aber auch möglich, das Verbindungsproblem weitgehend unabhängig vom funktionalen Problem zu lösen.

## 2.2 Unabhängiges Lösen der Teilprobleme

Mit der CIP-Methode wird das funktionale Verhalten eingebetteter Systeme durch CIP-Modelle formal spezifiziert. CIP-Modelle stellen kooperierende Zustandsmaschinen dar und werden mit CIP Tool<sup>®</sup> grafisch konstruiert. Für die Implementation werden CIP-Modelle in CIP-Units aufgeteilt, die durch einen Code-Generator automatisch in parallel ausführbare Softwarekomponenten transformiert werden.

Das Verbindungsproblem hingegen muss mit Werkzeugen angegangen werden, die der angewandten Interfacetechnologie angepasst sind.

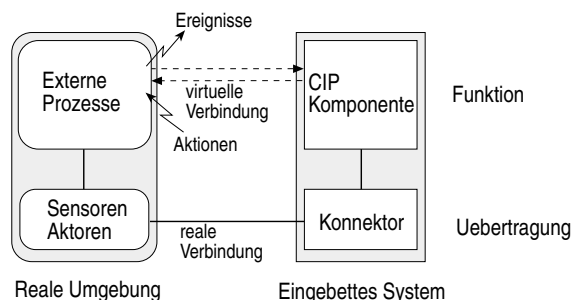


Bild 2 – Konzeptuelle Software-Architektur

Die Problemzerlegung führt zu entsprechenden Schichten in der Software-Architektur der entwickelten Lösung. Eine CIP-Komponente reagiert wie im CIP-Modell spezifiziert auf Ereignisse der Umgebung und erzeugt die notwendigen Aktionen. Die Verbindung zwischen CIP-Komponente und Umgebung ist virtuell und muss mittels Sensoren und Aktoren und entsprechenden Konnektoren (IO-Module) realisiert werden. Das Spezifizieren der virtuellen Verbindung stellt einen wesentlichen Entwicklungsschritt der CIP-Methode dar, weil damit die funktionale Abstraktionsebene festgelegt wird. Durch frühes Festlegen der virtuellen Verbindung können die Schnittstellen zwischen CIP-Komponenten und Konnektoren stabilisiert werden, was eine parallele Entwicklung dieser Systemteile ermöglicht.

Die vollständige Separierung der funktionalen und des Verbindungsproblems bringt grosse Vorteile im Entwicklungsprozess, da die beiden Aufgaben unabhängig voneinander gelöst werden können. Es handelt sich hier nicht einfach um die Zerlegung eines grossen Problems in zwei kleinere, sondern es wird die Entflechtung zweier Problemkreise erreicht, welche verschiedenen Abstraktionsebenen angehören.

### 3. CIP-Modelle

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 mathema-

tische 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.

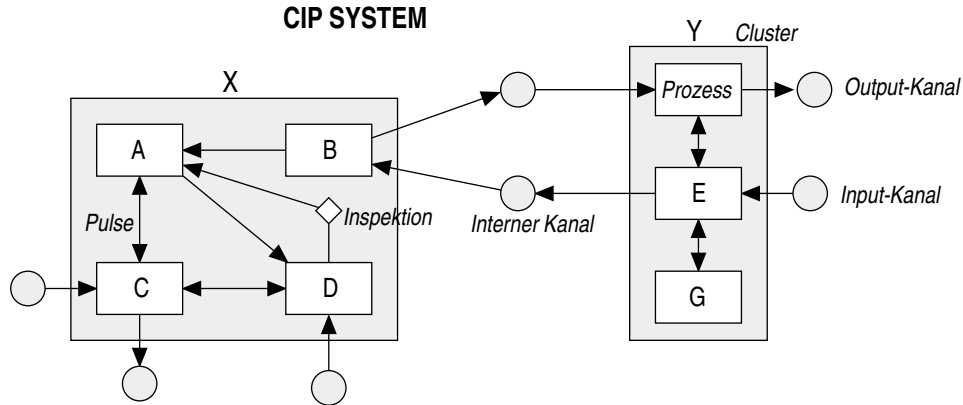


Bild 3 – CIP-Modell mit zwei Clustern

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 Kettenreaktion ist nicht unterbrechbar und definiert einen einzigen Zustandsübergang des ganzen Clusters. Der Output des Clusters besteht aus Meldungen, die von den aktivierten Prozesse in die entsprechenden Kanäle abgesetzt worden sind. Der synchrone Pulsübertragungsmechanismus wird als Interaktion, die asynchrone Übertragung von Meldungen als Kommunikation bezeichnet.

Verschiedene Modellierungssichten (Architektur, Verhalten, Kontrollfluss) werden automatisch konsistent gehalten. Die komponentenbasierte Konstruktionstechnik bietet wesentlich mehr Flexibilität bei Änderungen als herkömmliche Techniken mit formalen Spezifikations-sprachen. Ein Codegenerator transformiert Modelle in ausführbare Software-Komponenten. Auch strukturierte grafische Dokumentation wird automatisch erzeugt. Die Modellbeschreibungen in diesem Artikel sind Beispiele automatisch erzeugter Dokumentation.

CIP-Modelle werden mit CIP Tool® [1] konstruiert, indem grafische und textuelle Modellelemente miteinander verknüpft werden.

#### 3.1 PROCESSES – Erweiterte Zustandsmaschinen

Prozesse werden als erweiterte Zustandsmaschinen modelliert. Die Transitionsstrukturen und die in den Transitionen ausgeführten Operationen gehören zu zwei verschiedenen Abstraktionsebenen. Die Transitionsstruktur beschreibt die Kooperation mit anderen Prozessen des Systems und der Umgebung, während mit Operationen die lokale algorithmische Funktionalität definiert wird.

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 Outputpulsen gebildet.

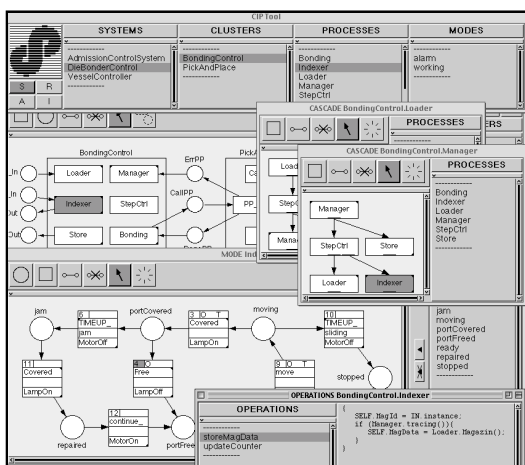


Bild 4 – Editoren von CIP Tool®

Im folgenden Bild ist die Schnittstelle des Prozesses *Door* beschrieben und sein Verhalten durch eine Transitionsstruktur spezifiziert.

PROCESS Door

INPORT DoorEvents  
 MESSAGES Closed, Opened  
 OUTPORT DoorActions  
 MESSAGES MotClose, MotOpen, MotOff  
 IMPULSES close, open  
 OUTPUTSES clsAck, opnAck

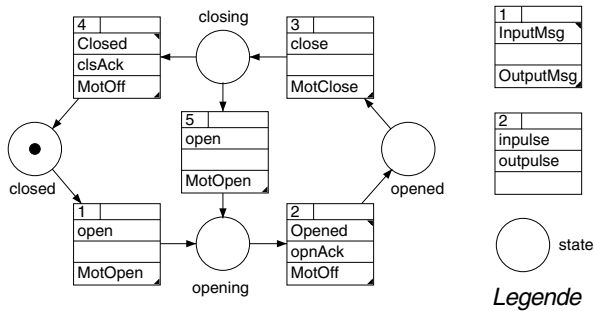


Bild 5 – Reine Zustandsmaschine

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 Output abgesetzt werden, der von mehreren Prozessen des Clusters empfangen werden kann (multi-cast). Zusätzlich kann in jeden Output 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.

PROCESS Cashier

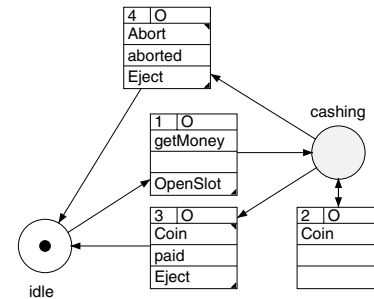
VARIABLES amount: int, price: int  
 INPORT EventPort  
 MESSAGES Coin; int, Abort  
 OUTPORT ActionPoert  
 MESSAGES Eject: int, OpenSlot  
 IMPULSES getMoney: int  
 OUTPUTSES aborted, paid: int

OPERATIONS

```
init {SELF.amount = 0; SELF.price = IN;}
incr {SELF.amount = SELF.amount + IN;}
...
```

CONDITIONS

notEnough (SELF.amount < SELF.price)



CONDITION ALLOCATION

2 notEnough, 3 ELSE\_

OPERATION ALLOCATION

1 init, 2 incr, 3 change, report, 4 ...

Bild 6 - Erweiterte Zustandsmaschine

Die Inputmeldung *Coin* des Prozesses *Cashier* zum Beispiel trägt einen Zahlenwert entsprechend dem Wert der eingeworfenen Münze. Mit Operationen wird der eingeworfene Betrag aufsummiert oder das Rückgeld berechnet. Wenn sich der Prozess im grau markierten Zustand *cashing* befindet, gibt es zwei mögliche Transitionen für die Inputmeldung *Coin* (Nondeterminismus). Durch entsprechend zugeordnete Bedingungen wird das Prozessverhalten eindeutig. Bedingungen können von den Daten des Inputs und den Werten der eigenen Variablen sowie von den Zuständen und Variablen anderer Prozesse desselben Clusters abhängen (siehe Zustandsinspektion).

Variablen, Datentypen, Operationen und Bedingungen werden in der generierten Programmiersprache formuliert. Die Code-Konstrukte sind Primitive und werden im generierten Code 'inline' eingebunden. Vom theoretischen Standpunkt aus wäre es schöner, eine eigene funktionale Sprache zu verwenden. In der Praxis hat sich jedoch der pragmatische Ansatz mit der Implementationssprache bewährt, erlaubt er doch problemlos Funktionen, Datentypen und Objektklassen bestehender Bibliotheken zu verwenden.

Die beiden Prozesse *Door* und *Cashier* sind Prozesse, die auf Meldungen reagieren, die aufgrund diskreter Zustandsänderungen eines

externen Prozess erzeugt worden sind. Eingebettete Systeme haben jedoch im allgemeinen einen hybriden Charakter, d. h. die Umgebung besteht sowohl aus diskreten und kontinuierlichen Prozessen. Die Überwachung und Steuerung kontinuierlicher Prozesse erfolgt in CIP durch Abtastmeldungen und Stellaktionen mit Daten. Die regeltechnischen Algorithmen werden in Operationen der aktivierten Transitionen ausgeführt.

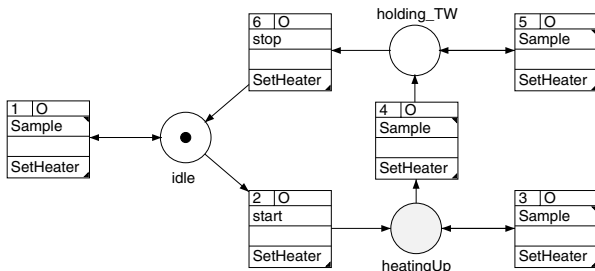


Bild 7 – Zeitgetriebene Zustandsmaschine (Sampling)

Bild 7 zeigt die Transitionsstruktur eines Prozesses, der die Temperatur einer Flüssigkeit mit einer kontinuierlich steuerbaren Heizung regelt. Der Prozess reagiert auf die periodisch auftretende *Sample* Meldung, welche die erfasste Temperatur überträgt. Den Transitionen sind Operationen mit Regelalgorithmen zugeordnet. Die neuen Werte für die Stellgrößen werden mit der *SetHeater* Outputmeldung übermittelt.

### 3.2 INTERACTION – Synchroner Pulsübertragung

Ein als INTERACTION NET bezeichneter Graph (Bild 8) definiert, zwischen welchen Prozessen Pulse übertragen werden. Für jede Interaktionsverbindung wird in einer entsprechenden Tabelle definiert, welche Outputpulse dem entsprechenden Empfänger gesendet werden und welcher Input dabei getriggert wird (Pulse Translation). Ein Output kann i. a. von mehreren Prozessen empfangen werden (Multicast).

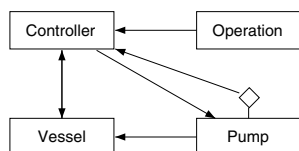


Bild 8 – INTERACTION NET eines Clusters

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 Output 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 Input getriggert wird.

### 3.3 STATE INSPECTION – Lesen fremder Prozessvariablen

Wie bereits erwähnt, können die Bedingungen einer Transitionsstruktur 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 (INQUIRY) des inspizierten Prozesses. Im Gegensatz zur Pulsübertragung, bei welcher sowohl Sender wie Empfänger aktiv an der Übertragung teilnehmen, ist bei einer Inspektion der inspizierte Prozess nicht aktiv. Durch Zustandsinspektion entsteht eine weitere Abhängigkeit zwischen Prozessen, die im INTERACTION NET durch Verbindungen mit Rhomben grafisch deklariert werden (Bild 8). Der Pfeil zeigt zum inspizierenden Prozess (Datenfluss).

### 3.4 COMMUNICATION – Asynchrone Kommunikation

Die Prozesse der verschiedenen Cluster kommunizieren untereinander und mit der Umge-

bung über asynchrone Kanäle. Asynchrone Kommunikation bedeutet im CIP-Modell, dass die Schreib- und die Leseaktion einer Übertragung zeitlich getrennt in verschiedenen Clustertransitionen ausgeführt werden. Input- und Outputkanäle bilden die Schnittstelle zur Umgebung, interne Kanäle sind Teil des CIP-Modells. Kanäle werden im grafischen COMMUNICATION NET mit den Ports der kommu-

nizierenden Prozesse verbunden. Die Kanäle modellieren ein aktives Kommunikationsmedium, welches die sequentielle Ordnung der übertragenen Meldungen erhält. Prozesse sind rezeptive Entitäten, welche immer bereit sind, Meldungen anzunehmen. Die notwendige Puffergröße der Kanäle ist nur durch den Durchsatz des implementierten Systems bestimmt und wird nicht im CIP-Modell spezifiziert.

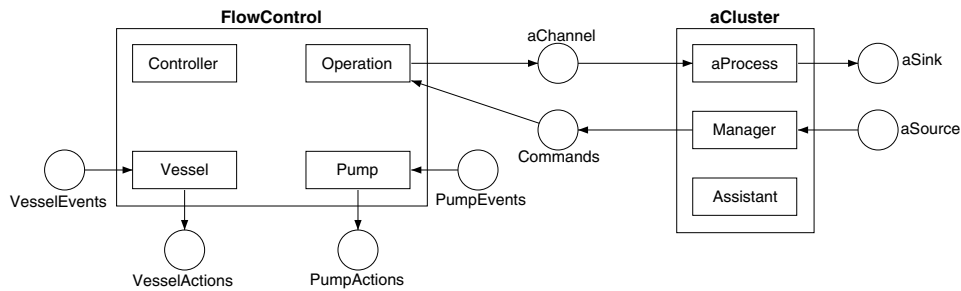


Bild 9 – COMMUNICATION NET eines CIP Systems

### 3.5 PROCESS ARRAYS – Replizierte Prozesse

Mehrfach auftretende Prozessinstanzen werden als mehrdimensionale Prozessarrays spezifiziert. Die Vielfachheiten der einzelnen Array-Dimensionen werden mittels Index-Typen definiert. Durch die Verwendung gemeinsamer Index-Typen in Prozess-Arrays können formal "Entity Relationship"-Strukturen definiert werden.

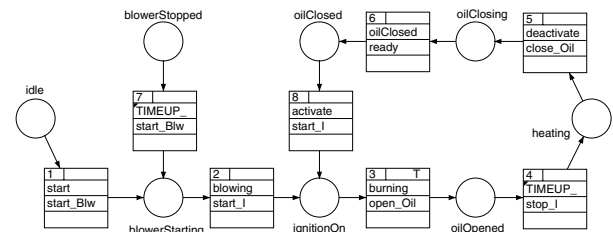
### 3.6 MASTER-SLAVE HIERARCHY – Verhaltensstrukturierung

Die Transitionsstruktur eines CIP-Prozesses beschreibt, wie der Prozess mit Zustandsübergängen und erzeugten Outputpulsen 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 müssen. Ein Alarmmeldung kann zum Beispiel dazu führen, dass der Prozess sich anders verhält bis der Alarm wieder zurückgesetzt wird. Die resultierende Transitionsstruktur stellt dann eine Superposition von zwei Strukturen dar, eine für den Normal- und eine für den Alarmfall.

Um solche Superpositionen zu entflechten wird das Verhalten des Prozesses durch alternative

Modi spezifiziert [2]. Jeder Modus wird grafisch durch eine Transitionsstruktur beschrieben, welche die den Zustände, Meldungen und Pulse des Prozesses verknüpft.

MODE aktiv



MODE deaktiviert

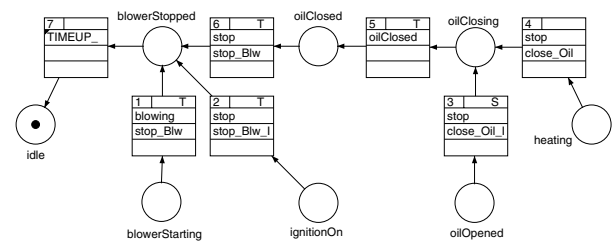


Bild 10 – Zwei Modi des Prozesses Controller einer Heizungssteuerung

Ein Moduswechsel kann durch einen oder mehrere als Master bezeichnete Prozesse bewirkt werden. Die Master-Slave Beziehungen eines Clusters werden wie in Bild 11 in einem entsprechenden Master-Slave Graphen definiert. Master-Slave Beziehungen werden durch Dreieck-Symbole dargestellt, die oben mit Master-

Prozessen und unten mit dem Slave-Prozess verbunden sind. Der gerichtete Graph muss immer azyklisch bleiben, damit die allgemeine Hierarchiebedingung erfüllt bleibt.

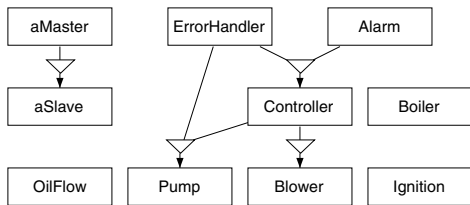


Bild 11 – MASTER-SLAVE Hierarchy graph

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 damit immer dann auftreten, wenn ein Master seinen Zustand wechselt.

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

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 bleibt damit erhalten. Dies wird besonders wichtig, wenn der aktuelle Prozesszustand ein Abbild eines Zustandes der Umgebung ist.

#### 4. Umgebungsorientierter Entwicklungsansatz

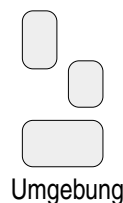
Mit formalen Verhaltensmodellen wird automatisch in einer definierten Abstraktionsebene gearbeitet. Zusätzlich zu solchen "Leitplanken" ist ein konzeptuelles Modell für den Modellierungsprozess notwendig, damit in nachvollziehbaren Schritten verständliche Lösungen gefunden werden können. Bei der Entwicklung eines CIP-Modells wird vom Verhalten der

Prozesse der Umgebung ausgegangen. Der Ansatz stammt aus der JSD-Methode (Jackson System Development) [3] und ist für Prozesssteuerungen wie geschaffen. Das umgebungsorientierte Vorgehen führt einerseits zu verständlichen Modellen, was sich unmittelbar auf Qualitätsmerkmale wie Zuverlässigkeit, Wartbarkeit und Änderungsfreundlichkeit auswirkt. Andererseits erlaubt dieser Ansatz, das funktionale und das Verbindungsproblem unabhängig voneinander zu lösen.

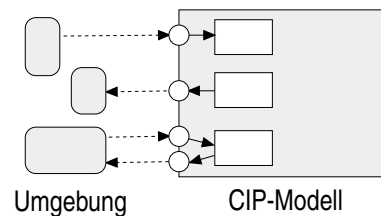
CIP-Modelle werden in drei Entwicklungsschritten konstruiert:

1. Ereignisse und Aktionen bezeichnen
2. Entwicklung eines Kontextmodells
3. Konstruktion der Steuerfunktionen

##### 1. Ereignisse & Aktionen



##### 2. Kontext-Modell



##### 3. Vollständiges Modell

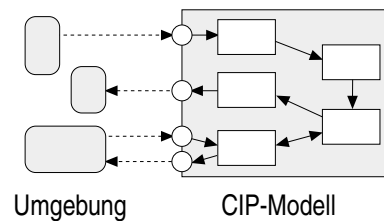


Bild 12 – Entwicklungsschritte

#### 4.1 Ereignisse und Aktionen

In einem ersten Schritt wird mittels Ereignis- und Aktionenlisten eine virtuelle Schnittstelle zur realen Umgebung festgelegt. Ereignisse

werden durch die Umgebung, Aktionen durch das eingebettete System erzeugt. Beides sind reale Phänomene, die für die Beschreibung des funktionalen Verhaltens wesentlich sind und nicht von der angewandten Interface-Technologie abhängen.

Entsprechende Ereignis- und Aktionsmeldungen definieren die Input- und Output-Schnittstelle auf der CIP-Seite der virtuellen Verbindung. Attribute von Ereignissen und Aktionen werden als Daten der entsprechenden Meldungen übertragen. Ein Ereignisattribut beschreibt einen Umstand, der beim Auftreten des Ereignisses zutrifft, wie zum Beispiel der erfasste Strich-Code eines Scanner-Ereignisses oder die Parameter eines Bediener-Befehls. Ein Aktionsattribut hingegen beschreibt einen Sachverhalt, für den beim Erzeugen der Aktion gesorgt werden muss, wie etwa die verlangte Durchflussrate eines geöffneten Ventils.

Ein Ereignis wird entweder als Prozess- oder Zeitereignis klassifiziert. Prozessereignisse werden durch die Eigendynamik eines externen Prozesses erzeugt und treten zu nicht vorausbestimmbaren Zeitpunkten auf. Solche Ereignisse bezeichnen meistens diskrete Zustandsübergänge externer Prozesse. Diskrete Zustände stellen oft Abstraktionen dar, die eine ganze Menge von physikalischen Zuständen beinhalten, wie zum Beispiel die Niveaubereiche eines Kessels oder der Ready-Zustand eines technischen Gerätes.

Kontinuierliche Zustandsänderungen werden wie üblich durch Abtastung erfasst. Abtastereignisse sind periodische Zeitereignisse, deren Attribute die erfassten Prozesszustände beschreiben. Der Einfluss auf kontinuierliche Prozesse erfolgt entsprechend durch periodisch abgesetzte Aktionsmeldungen, welche die Stellgrößen übermitteln.

Das Erarbeiten der Ereignis- und Aktionslisten stellt einen entscheidenden Entwicklungsschritt dar, weil damit die Abstraktionsebene für die funktionale Problemlösung festgelegt wird. Die spezifizierten Ereignisse und Aktionen müssen einerseits erlauben ein CIP-Modell zu konstruieren, welches den funktionalen Anforderungen genügt. Andererseits muss auch sichergestellt werden, dass alle

Ereignisse erfasst und alle Aktionen erzeugt werden können.

Wenn die virtuelle Schnittstelle festgelegt ist, können sowohl das funktionale wie auch das reale Verbindungsproblem unabhängig voneinander gelöst werden. Iterationschritte im Entwicklungsprozess tangieren die gesamte Problemlösung nur noch, wenn die virtuelle Verbindung geändert werden muss.

## **4.2 Kontextmodell: Modellprozesse**

Das Kontextmodell wird aufgrund von Beschreibungen und Prozessmodellen der Umgebung erstellt. Es besteht aus einer Menge von unabhängigen CIP-Prozessen, die als Modellprozesse bezeichnet werden. Modellprozesse empfangen Ereignismeldungen von Inputkanälen und erzeugen Aktionsmeldungen über entsprechende Outputkanäle. Die Transitionsstrukturen der Modellprozesse definieren die gültigen Folgen empfangener und erzeugter Meldungen und haben damit den Stellenwert sequentieller Kommunikationsprotokolle. Das Kontextmodell beschreibt somit das mögliche Verhalten der einzelnen externen Prozesse aus der Sicht des zu entwickelnden Systems. Die Modellprozesse bilden so die Grundlage für die Entwicklung der reaktiven Funktionalität des Systems.

## **4.3 Steuerfunktionen: Interaktion, interne Kommunikation**

In weiteren Entwicklungsphasen wird das Gesamtverhalten des eingebetteten Systems spezifiziert. Dazu werden Funktionsprozesse eingeführt und mittels Interaktion und interner Kommunikation die notwendigen Abhängigkeiten zwischen den Modellprozessen erzeugt. In einem ersten Schritt entwickelt man die primäre Funktionalität des Systems, die sich auf das durch die Modellprozesse definierte Normalverhalten abstützt. Damit auch auf unerwartetes Verhalten der Umgebung und auf Übertragungsfehler reagiert werden kann, müssen meistens in weiteren Schritten die Strukturen der Modellprozesse erweitert und Fehlerprozesse eingeführt werden. Wie die Erfahrung zeigt, gehört die Spezifikation des Systemverhaltens für den Fehlerfall zu den schwierigsten Problemen bei der Entwicklung robuster und sicherer Systeme. Ohne explizites Modell

des Normalverhaltens kann diese Aufgabe meist gar nicht befriedigend gelöst werden.

## 5. Komponentenorientierte Implementation

Für die Implementation eines CIP-Modells werden die Cluster zu CIP Units gruppiert. Jede CIP Unit wird durch den Code-Generator automatisch in eine parallel ausführbare Software-Komponente transformiert. Der erzeugte Code besteht aus zwei separat generierten Teilen, der CIP Shell und der CIP Machine.

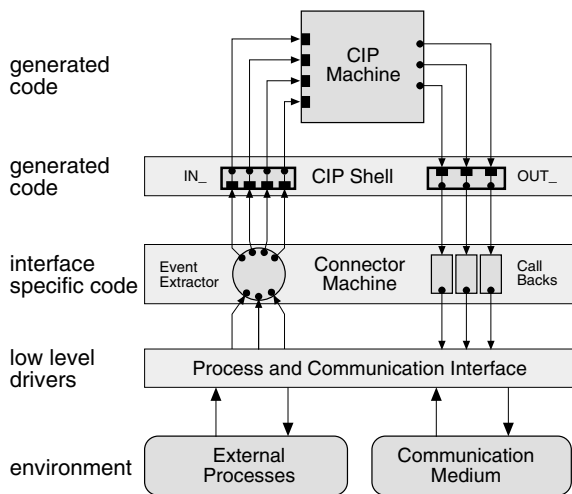


Bild 13 – Implementation einer CIP Unit

Die *CIP Shell* ist die Schnittstelle der Komponente und besteht aus zwei linearen Strukturen von Funktionspointern, entsprechend den Input- und Outputkanälen der CIP Unit. Der Code einer CIP Shell wird aus den entsprechenden Kanalmodellen generiert und ist damit vollkommen unabhängig von den Cluster-Modellen.

Die *CIP Machine* implementiert das reaktive Verhalten der CIP Unit und wird über die CIP Shell durch Aufrufe von Input-Kanalfunktionen aktiviert. Jeder Aufruf führt zu einer Clustertransition, aus der über die CIP Shell Output-Kanalfunktionen aktiviert werden (Call Back).

Eine CIP Unit muss mit einem Teil der externen Prozesse verbunden werden. Dieser Teil der Implementation ist stark durch die an der Prozessperipherie angewandte Technologie bestimmt und stellt meistens ein Problem für sich dar. Die Partitionierung eines CIP-Modells erzeugt aber auch neue Verbindungsprobleme wegen der durch die Aufteilung aufgebroche-

nen Kanäle. Die entsprechende Kommunikation zwischen CIP Units kann jedoch meistens mit standardisierten Kommunikationstechniken gelöst werden (Feldbusse, serielle Kommunikation, Sockets).

Aufgrund der in CIP-Modellen modellierten Kooperation von Clustern und Prozessen entfällt die übliche Implementation konzeptueller Parallelität mittels Multitasking vollständig. Meistens wird pro Prozessor nur eine Unit implementiert. Task Scheduling und Interrupt Handling wird damit auf Hardware-Schnittstellenfunktionen und mögliche Hintergrunddienste beschränkt [4].

Der Umgebungsorientierte Entwicklungsprozess erlaubt wie bereits erwähnt die Schnittstellen der CIP-Modelle zu einem sehr frühen Zeitpunkt festzulegen. Das Modellierungswerkzeug erlaubt zudem, solche Schnittstellen über längere Entwicklungsperioden zu sperren (locking). Der generierte CIP Shell Code bekommt damit eine Bedeutung als stabilisierbares Verbindungselement zwischen parallel entwickelten CIP-Modellen und Implementationsmodulen. Das Konzept ist erfolgreich in industriellen und akademischen Projekten angewendet worden. Verschiedene Partitionen desselben CIP Modells wurden in den unterschiedlichen Validierungsphasen sowohl mit Simulationsmodellen, Testumgebungen, Prüfständen wie auch mit der realen Prozessumgebung verbunden. Bei der Entwicklung eines Autos mit Hybridantrieb konnte selbst der Code für die Implementationsmodule aus formalen Beschreibungen generiert werden [5].

## 6. Fall-Beispiel: TransportControlSystem

Das Beispiel zeigt, wie ein CIP-Modell entwickelt wird und wie eine einfache Master-Slave-Hierarchie funktioniert. Der resultierende Cluster stellt eine formal ausführbare Lösung des gestellten Problems dar. Weil die algorithmischen Anforderungen des Beispiels trivial sind, besteht das spezifizierte CIP-Modell nur aus nicht-erweitererten Zustandsmaschinen.

### 6.1 Problemstellung

#### Beschreibung der Anlage

Die Anlage besteht aus einem Förderer, der Objekte in eine Richtung transportiert, einem Scanner, mit welchem aufgelegte Objekte identifiziert werden können und einem Schalter als Bedienelement.

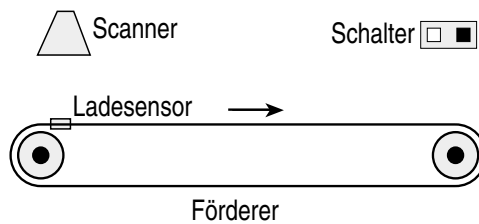


Bild 14 – Anlage

#### Anforderungen

Das System wird aktiv, sobald das erste Objekt auf den Förderer gelegt wird. Wenn der Förderer während 30 Sekunden nicht neu beschickt wird, schaltet der Motor des Förderers aus. Es sollen zwei verschiedene Betriebsarten realisiert werden: Wenn der Schalter ausgeschaltet ist, transportiert der Förderer kontinuierlich aufgelegte Objekte. Bei eingeschaltetem Schalter wird der Förderer bei jeder Beschickung angehalten und der Scanner aktiviert. Der Förderer beginnt wieder zu laufen, sobald der Scanning Vorgang abgeschlossen worden ist.

### 6.2 Lösung

Im Folgenden werden die Resultate der drei Entwicklungsschritte der CIP-Methode dargestellt.

#### A. Virtuelle Schnittstelle: Ereignisse und Aktionen

Im ersten Entwicklungsschritt werden zum einen Ereignisse definiert, auf die im CIP-Modell reagiert wird, zum anderen Aktionen festgelegt, die aus dem CIP-Modell erzeugt werden.

##### Ereignisse

On / Off	Der Schalter wird ein / aus geschaltet.
Load / Free	Ein Objekt wird aufgeladen / hat sich vom Ladeplatz wegbewegt.
Scanned	Der Scanning-Vorgang ist abgeschlossen.

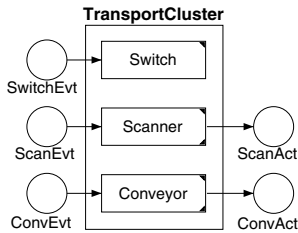
##### Aktionen

MotOn / MotOff	Ein- / Ausschalten des Motors des Förderers
Scan	Aktivierung des Scanners

## B. Kontextmodell: Kanäle und Modell-Prozesse

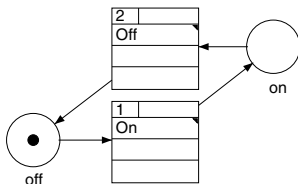
Als erste Prozesse des CIP-Modells werden Modellprozesse spezifiziert, die das Verhalten der externen Prozesse aus der Sicht der zu entwickelnden Steuerung modellieren. Diese Prozesse werden mit entsprechenden Input- und Outputkanälen verbunden, aus denen sich bereits der Code für die Schnittstelle der Implementation generieren lässt. Die Transitionsstrukturen dieser noch unvollständigen und unabhängigen Prozesse stellen Kommunikationsprotokolle mit der Umgebung dar. Der Prozess Conveyor (Förderer) wurde bereits mit zwei Modi ausgestattet, die den angeforderten Betriebsarten entsprechen.

### COMMUNICATION NET InterfaceChannels

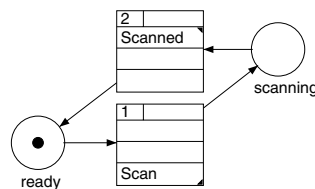


CHANNEL SwitchEvt MESSAGES Off, On  
 CHANNEL ScanEvt MESSAGES Scanned  
 CHANNEL ScanAct MESSAGES Scan  
 CHANNEL ConvEvt MESSAGES Free, Load  
 CHANNEL ConvAct MESSAGES MotOff, MotOn

### PROCESS Switch

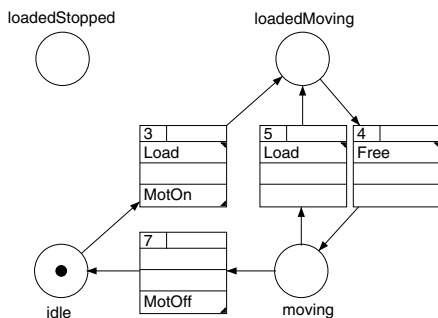


### PROCESS Scanner

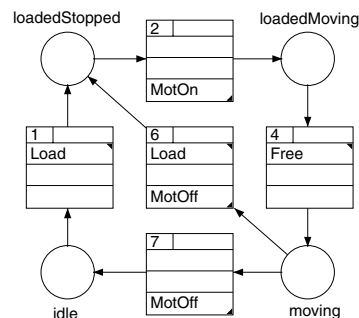


### PROCESS Conveyor

#### MODE ongoing



#### MODE stepped

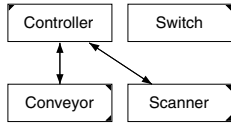


*Legende:* Die Eckmarkierungen bei Prozessboxen zeigen an, ob der Prozess externen Input bzw. Output hat. Obere Ecke rechts: Kanalinput; Obere Ecke links: Timerinput; Untere Ecke rechts: Kanaloutput. Analoge Markierung treten bei den Transitionsboxen auf.

## C. Vollständiges CIP-Modell

Der neu eingeführte Funktionsprozess Controller steuert die Kooperation des Conveyor- und des Scanner-Prozesses. Auch der Controller hat den Betriebsarten entsprechend zwei Modi. Der aktive Modus des Controller- und des Conveyor-Prozesses ist jeweils durch den aktuellen Zustand des Switch-Prozesses bestimmt (MODE SETTING). Der Controller-Prozess ist zusätzlich für die Zeitüberwachung zuständig und wurde deshalb mit einem Timer ausgestattet (Modelling Sugar von CIP Tool).

### INTERACTION NET

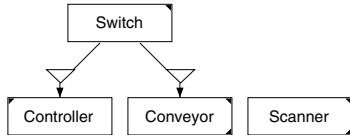


### PULSE TRANSLATIONS

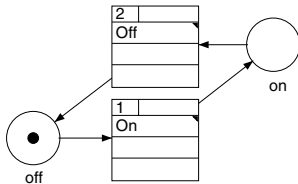
- Controller.move -> Conveyor.move
- Controller.stop -> Conveyor.stop
- Controller.scan -> Scanner.scan
- Conveyor.loaded -> Controller.loaded
- Scanner.scanned -> Controller.scanned

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

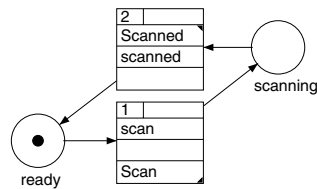
### MASTER-SLAVE GRAPH



### PROCESS Switch



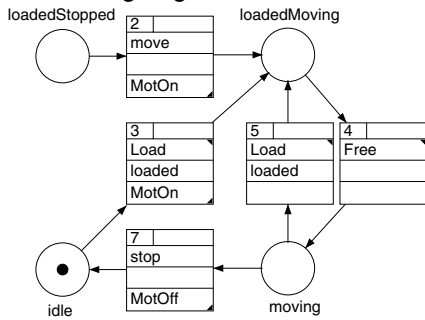
### PROCESS Scanner



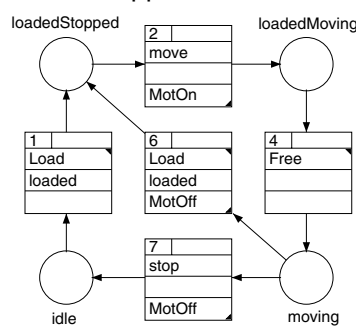
### PROCESS Conveyor

MODE SETTING Switch.off -> ongoing, Switch.on -> stepped

### MODE ongoing



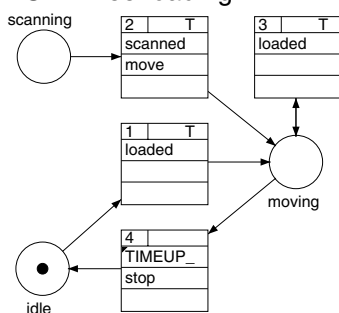
### MODE stepped



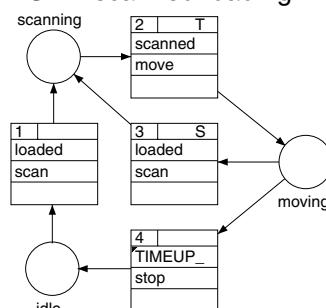
### PROCESS Controller

MODE SETTING Switch.off -> freeLoading, Switch.on -> scannedLoading

### MODE freeLoading



### MODE scannedLoading



### Legende



Timer wird gesetzt



Timer wird gestoppt

## Animation

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 *Load, On, Free, Load, Scanned* erzeugte Animationsgeschichte (Trace). Jede Tabelle beschreibt eine Clustertransition:

PROCESS	MODE	PRESTATE	INPUT	POSTSTATE	OUTPUT
Conveyor	ongoing	idle	<b>Load</b>	loadedMoving	loaded, MotOn
Controller	freeLoading	idle	loaded	moving	SET TIMER

Switch	-	off	<b>On</b>	on	
Conveyor	stepped				
Controller	scannedLoading				

Conveyor	stepped	loadedMoving	<b>Free</b>	moving	
----------	---------	--------------	-------------	--------	--

Conveyor	stepped	moving	<b>Load</b>	loadedStopped	loaded, MotOff
Controller	scannedLoading	moving	loaded	scanning	scan, STOP TIMER
Scanner	-	ready	scan	scanning	Scan

Scanner	-	scanning	<b>Scanned</b>	ready	scanned
Controller	scannedLoading	scanning	scanned	moving	move , SET TIMER
Conveyor	stepped	loadedStopped	move	loadedMoving	MotOn

## 7. Zusammenfassung

Die CIP-Methode wird bei der Entwicklung von eingebetteten Systemen eingesetzt, die komplexe technische Geräte oder Anlagen steuern. Das ingenieurmässige Arbeiten wird durch Entwicklungskonzepte und Modellierungstechniken unterstützt, die der speziellen Problemstellung angepasst sind.

Bei der Entwicklung eines eingebetteten Systems sind grundsätzlich zwei Aufgaben zu lösen: Einerseits muss dafür gesorgt werden, dass das System funktional richtig auf Ereignisse der Umgebung reagiert, andererseits muss an der Prozessperipherie Information erfasst und erzeugt werden. Das allgemeine Entwicklungsproblem wird daher in ein funktionales und ein Verbindungsproblem aufgeteilt. Das funktionale Problem wird durch Konstruktion eines formalen Verhaltensmodell mit CIP Tool®

[1] gelöst, das Verbindungsproblem mit Techniken, die der Sensor/Aktor-Technologie angepasst sind. Der vorgeschlagene Entwicklungsprozess erlaubt, die beiden Probleme unabhängig voneinander zu lösen.

Die meisten reaktiven Verhaltensmodelle basieren entweder auf synchroner oder asynchroner Kooperation von Prozessen. CIP kombiniert beide Kooperationsparadigmen im selben Modell. CIP-Modelle bestehen aus nebenläufigen Clustern, die aus synchron kooperierenden Zustandsmaschinen zusammengesetzt sind. Synchrone Kooperation ist notwendig, damit nicht-unterbrechbare interne Interaktionspropagation modelliert werden kann. Asynchrone kooperierende Cluster hingegen bilden parallele modulare Blöcke und ermöglichen, CIP-Modelle verteilt zu implementieren.

CIP-Modelle werden mittels architekturbasierter Komposition konstruiert, indem Interaktions- und Kommunikationsverbindungen explizit in entsprechenden grafischen Modellen erzeugt werden. Unbeschränkte zyklische Reaktionsketten werden automatisch durch das Tool verhindert. Ein Interactiontree-Viewer erlaubt zudem, bereits während dem Modellieren alle möglichen Interaktionssequenzen als graphische Baumstrukturen darzustellen.

Verhaltensstrukturierung wird durch Master-Slave-Hierarchen [2] ermöglicht. Das neuartige Hierarchiemodell basiert auf Master-Prozessen, die Verhaltensänderungen in Slave-Prozessen bewirken können. Die damit erzeugte hierarchische Struktur erweist sich als wesentlich flexibler als starre Kompositionshierarchien, die auf der Verschachtelung von Transitionsstrukturen beruhen.

Reaktive Steuerfunktionen müssen ständig das aktuelle Verhalten der gesteuerten Prozesse reflektieren. Der Modellierungsprozess der CIP-Methode beginnt daher, ähnlich wie in JSD [3], mit der Entwicklung eines Kontextmodells, welches die gültigen zeitlichen Folgen der Interaktionen mit der Umgebung beschreibt. Das funktionale Modell wird in weiteren Entwicklungsschritten erarbeitet, indem das Kontextmodell mit zusätzlich eingeführten Funktionsprozessen verknüpft wird.

Die Implementation von CIP-Modellen basiert auf Softwarekomponenten-Technologie. Verschiedene Konfigurationen automatisch erzeugter Softwarekomponenten können leicht mit IO-Modulen und anderen Softwarekomponenten des Systems verbunden werden [4, 5]. Die resultierende Flexibilität bei der Bildung ausführbarer Systemteile ist wichtig, da eingebettete Systeme normalerweise in verschiedenen Testumgebungen (Simulationsmodelle, Prüfstände) validiert werden müssen.

### Probeversion von CIP Tool®

Eine Probeversion des CIP-Werkzeuges und ein Tutorial mit einfachen Beispielen ist auf der CIP-Homepage verfügbar:

<http://www.ciptool.ch>.

## Literatur

1. CIP Tool® - User Manual (1998). CIP System AG. Solothurn, Switzerland. Internet: <http://www.ciptool.ch>
2. H. Fierz. "The CIP Method. Component- and Model-Based Construction of Embedded Systems". 7th European Software Engineering Conference/7th and ACM SIGSOFT Symposium on the Foundations of Software Engineering, Toulouse, France. LNCS Vol. 1687, Springer Verlag Berlin, pages 374-391, (1999)
3. Cameron J.R.(ed.): JSP and JSD: The Jackson Approach to Software Development. IEEE Computer Society Press, (1989)
4. Trutman HO.: Well-Behaved Applications Allow for More Efficient Scheduling. Submitted to 24th IFAC/IFIP Workshop on Real-Time Programming and the Third Int. Workshop on Active and Real-Time Database Systems. Schloss Dagstuhl, Saarland Nay (1999)
5. Trutmann HO.: Generation of Embedded Control Systems. 23rd IFAC/IFIP Workshop on Real Time Programming, WRTP 98, Shantou, P.R. China (1998) 99-104

## Kontakte

Dr. Hugo Fierz  
Institut für Technische Informatik und  
Kommunikationsnetze  
ETH-Zentrum  
CH-8092 Zürich  
Tel: +41 (1) 632 70 65  
Fax: +41 (1) 632 10 36  
E-Mail: [fierz@tik.ee.ethz.ch](mailto: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: [info@ciptool.ch](mailto:info@ciptool.ch)

Internet:  
<http://www.ciptool.ch>