

Automatische Interaktionsanalyse in CIP-Modellen

Hugo Fierz

Institut für Technische Informatik und Kommunikationsnetze, ETH Zürich

E-Mail: fierz@tik.ee.ethz.ch

Hansruedi Müller

CIP System AG, Zürich

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

1. Problemstellung

Reaktive Echtzeitsysteme treten vor allem bei eingebetteten Systemen zur Steuerung und Überwachung externer Prozesse auf. Die Aufgabe eines reaktiven Systems ist, auf externe Ereignisse zu reagieren und die externen Prozesse gezielt durch bestimmte Aktionen zu beeinflussen. Reaktive Systeme bestehen meistens aus kooperierenden Systemteilen wie Prozessen, Objekten oder Funktionen. Ein externes Ereignis bewirkt fast immer eine interne Interaktionssequenz, d. h. mehrere Systemteile werden über modellspezifische Interaktionsmechanismen wie Message Passing, Event Broadcast oder Funktionsaufrufe aktiviert.

Eine zentrale Schwierigkeit bei der Entwicklung reaktiver Systeme ist, die Übersicht über alle Interaktionssequenzen zu behalten. Weil solche dynamische Strukturen implizit definiert sind, müssen sie bei der Entwicklung ständig gedanklich nachvollzogen und schliesslich zur Laufzeit überprüft werden.

Zusätzliche Schwierigkeiten treten auf, wenn für ein externes Ereignis mehr als eine Interaktionssequenz möglich ist. Dieser implizite Non-Determinismus entsteht typischerweise, wenn das reaktive System aus nebenläufigen Komponenten besteht, wie z. B. in SDL [1] oder ROOM [2].

Äusserst problematisch sind zudem zyklische Interaktionspfade, die unbeschränkt lange Interaktionssequenzen ermöglichen, wie das zum Beispiel in Statecharts-Modellen [3] möglich ist. Angeforderte Antwortzeiten können in solchen Fällen nur durch aufwendiges Testen "garantiert" werden. Zyklische Aktivierungen sind an und für sich nicht problematisch, solange sichergestellt ist, dass die Länge der resultierenden Interaktionssequenzen beschränkt ist.

Die Beschreibung von Interaktionssequenzen ist durch die in UML [4] verwendeten *Collaboration Diagrams* bekannt geworden. Mit solchen Diagrammen können einzelne Kollaborationsszenarien von Objekten dargestellt werden. Im Unterschied zur hier präsentierten Interaktionsanalyse stellen solche Diagramme aber lediglich Entwurfsanforderungen dar, die keinen formalen Bezug zur konstruierten Software haben.

Die in diesem Beitrag vorgestellte Interaktionsanalyse der CIP-Methode erlaubt, bereits während der Konstruktion von CIP-Modellen die Gesamtheit der Übertragungssequenzen interner Ereignisse automatisch darzustellen.

2. CIP-Modelle

CIP [5] ist eine modellbasierte Entwicklungsmethode für Steuer- und Leitsysteme. CIP-Modelle bestehen aus formalen Architektur- und Verhaltensmodellen, welche synchron und asynchron kooperierende Zustandsmaschinen beschreiben. Für algorithmische Belange werden diese Zustandsmaschinen, die als Prozesse bezeichnet werden, mit lokalen Variablen, Operationen und Bedingungen erweitert. Verhaltensabstraktion wird durch modusbasierte Verhaltenshierarchien unterstützt [6]. CIP-Modelle werden mit CIP Tool® [7] konstruiert, ein auf die CIP-Methode zugeschnittenes Framework von Graphik- und Texteditoren. Ein Codegenerator erzeugt aus den grafischen Modellen ausführbare Softwarekomponenten, welche flexibel in Testumgebungen und auf den Zielsystemen eingebettet werden können.

Reaktive Subsysteme werden in CIP als *Cluster* bezeichnet. Jeder Cluster setzt sich aus synchron interagierenden Prozessen zusammen. Erzeugt eine externe Ereignis-Meldung einen Zustandsübergang eines Prozesses, kann der aktivierte Prozess durch ein als *Output* bezeichnetes internes Ereignis *Impulse* anderer Prozesse triggern (Multicast). Die aktivierten Prozesse können wiederum durch erzeugte Outputpulse weitere Prozesse anstoßen. Die entstehende Interaktionskette ist nicht unterbrechbar und definiert einen einzigen Zustandsübergang des ganzen Clusters.

Der Pulsfluss der interagierenden Prozesse wird in einem Interaktionsnetz (INTERACTION NET) durch gerichtete grafische Konnektoren definiert. Die von einem Prozess aus gehenden Konnektoren werden als total geordnete Menge spezifiziert (CAST ORDER). Diese Ordnung definiert die Aktivierungsreihenfolge der Empfänger, wenn ein Output gesendet wird. Für jeden Konnektor wird als sogenannter Glue eine PULSE TRANSLATION definiert, die Outputpulse des Senders mit Impulsen des Empfängers verknüpft. Die Pulse Translation definiert die Interaktion von Sender und Empfänger durch eine explizite funktionale Output/Input-Relation.

2.1. Einfaches Beispiel eines CIP-Modells: TwoVesselSystem

Die folgende vollständige Modell-Beschreibung entspricht einem von CIP Tool generierten System-Report.

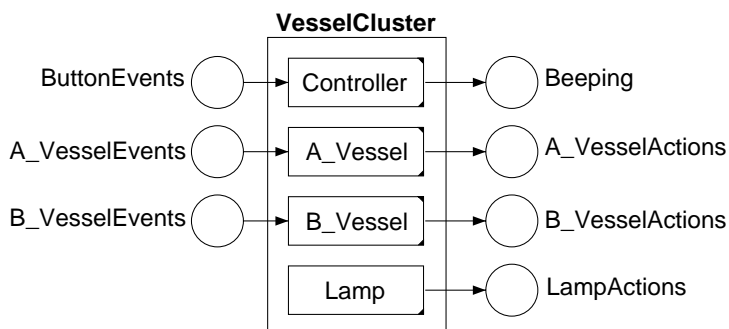
SYSTEM TwoVesselSystem

COMMENT

Steuerung zweier Kessel:

Wenn ein Bedienungsknopf gedrückt wird, muss Kessel A einmal gefüllt und geleert werden. Wird der Knopf erneut gedrückt bevor Kessel A geleert ist, muss auch Kessel B einmal gefüllt und geleert werden. Das Füllen und Leeren der Kessel wird durch eine brennende Lampe angezeigt.

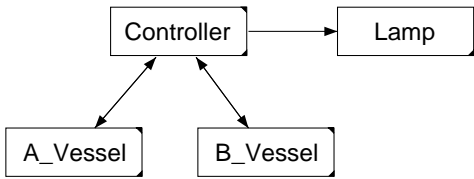
COMMUNICATION NET SourcesAndSinks



Das einfache Beispiel besteht aus einem einzigen Cluster. Das Kommunikationsnetz modelliert hier mit Input- und Output-Kanälen lediglich die Schnittstelle zur realen Prozessumgebung

CLUSTER VesselCluster

INTERACTION NET

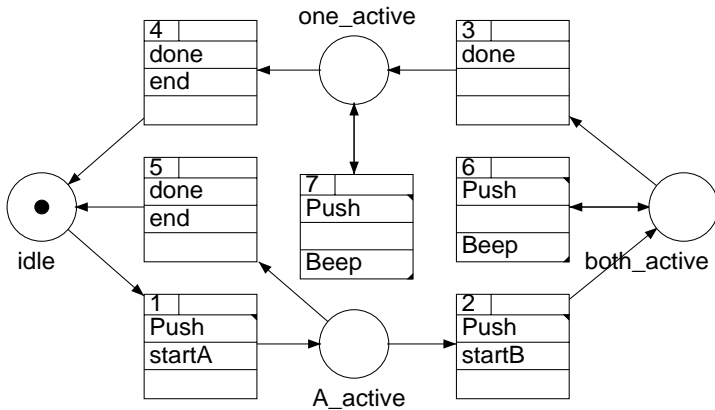


Das *Interaction Net* definiert als Ereignisflussgraph die möglichen Übertragungswege für Pulse zwischen den Prozessen eines Clusters. Ein Prozess kann denjenigen Prozessen Pulse senden, die mit einem entsprechenden grafischen Konnektor verbunden sind.

PROCESSES

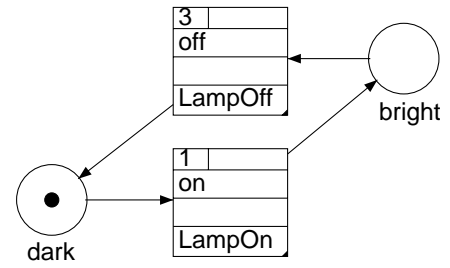
PROCESS Controller

INPORT CmdPort
 MESSAGES Push
 OUTPORT BeepPort
 MESSAGES Beep
 IMPULSES done
 OUTPUTS end, startA, startB



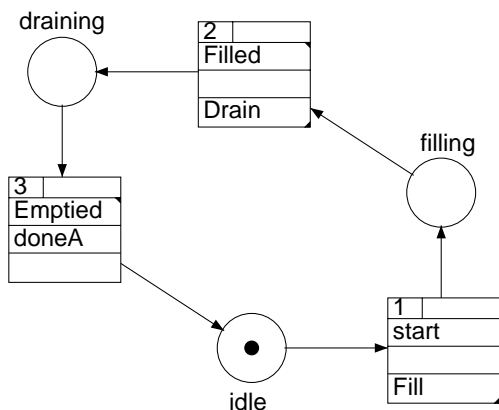
PROCESS Lamp

OUTPORT LmpPort
 MESSAGES LampOff, LampOn
 IMPULSES off, on



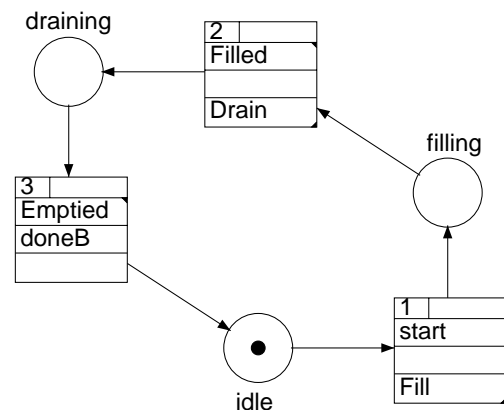
PROCESS A_Vessel

INPORT EventPort
 MESSAGES Emptied, Filled
 OUTPORT ActionPort
 MESSAGES Drain, Fill
 IMPULSES start
 OUTPUTS doneA

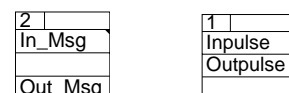


PROCESS B_Vessel

INPORT EventPort
 MESSAGES Emptied, Filled
 OUTPORT ActionPort
 MESSAGES Drain, Fill
 IMPULSES start
 OUTPUTS doneB



Legende:



CAST ORDER

```

SENDER Controller
  RECEIVERS A_Vessel, B_Vessel, Lamp
SENDER A_Vessel
  RECEIVERS Controller
SENDER B_Vessel
  RECEIVERS Controller

```

Ein Output eines Senders kann für mehrere Empfänger bestimmt sein. Damit solche *Multicast* nicht zu non-deterministischen Kontrollflüssen führen, wird die Pulsübertragung als *sequentieller Multicast* spezifiziert: Die Cast-Ordnung bestimmt die Aktivierungsreihenfolge der Empfänger.

In diesem Beispiel spielt die Cast-Ordnung keine Rolle. Sie wird für das Verhalten eines Clusters erst relevant, wenn zum Beispiel ein Prozess in derselben Clustertransition mehr als einmal aktiviert wird.

Die Funktionsweise der Steuerung ist leicht aus der Transitionstruktur des Prozesses *Controller* ersichtlich:

Das erste Ereignis *Push* bewirkt über den Output *startA* Zustandsübergänge in den Prozessen *A_Vessel* und *Lamp*, welche die externen Aktionen *Fill* bzw. *LampOn* erzeugen. Das zweite *Push*-Ereignis bewirkt analog über den Output *startB* das Füllen von Kessel B. Weitere *Push*-Ereignisse werden direkt mit *Beep* beantwortet. Wenn der Kessel A oder Kessel B leer geworden ist (Ereignis *Emptied*), wird beim Prozess *Controller* der Impuls *done* getriggert, was durch den Output *end* beim Prozess *Lamp* das Ausschalten der Lampe bewirkt, falls beide Kessel leer geworden sind.

3. Automatische Interaktionsanalyse

Nach einer allgemeinen Beschreibung des *Interaction Analysis Tool* werden die vom Werkzeug für das Fallbeispiel *TwoVesselSystem* dargestellten Interaktionssequenzen erläutert. Anschliessend folgt eine Betrachtung einiger Interaktionsbäume eines komplexeren Systems (*HomeHeatingSystem*).

3.1. Das Interaction Analysis Tool

Aus den formal spezifizierten Transitionstrukturen der Prozesse und den mittels Pulse Translation definierten Interaktionsrelationen können alle zur Laufzeit möglichen Interaktionssequenzen hergeleitet werden. Wiederholte Aktivierung desselben Prozesses in einer Interaktionssequenz ist erlaubt, solange diese Aktivierungen durch verschiedene Impulse des Prozesses ausgelöst werden. Eine wiederholte Aktivierung eines Prozesses mit demselben Impuls soll hingegen nicht möglich sein, da bei zyklischen Interaktionssequenzen mit unbeschränkten Antwortzeiten des Systems gerechnet werden muss.

Während der Modellkonstruktion unterhält CIP Tool ein Datenmodell eines gerichteten azyklischen Graphen, der alle Interaktionssequenzen enthält. Einerseits wird bei jedem Modellierungsschritt sichergestellt, dass keine zyklischen Interaktionssequenzen möglich werden. Andererseits erlaubt das Datenmodell zu jedem Zeitpunkt die möglichen Interaktionssequenzen zu visualisieren. Dazu selektiert der Anwender im *Prozess Interface Browser* eine Input-Meldung (externes Ereignis), einen Impuls oder einen Output eines beliebigen Prozesses. Im offenen *Interaction Analysis Window* wird dann der durch den Input, bzw. Output, bewirkte Interaktionsbaum grafisch dargestellt. Die Ausführung eines solchen Interaktionsbaumes ist als Linkstraversierung des Baumes zu lesen, wobei bei

PULSE TRANSLATIONS

```

SENDER Controller
  startA  -> A_Vessel.start, Lamp.on
  startB  -> B_Vessel.start
  end     -> Lamp.off
SENDER A_Vessel
  doneA   -> Controller.done
SENDER B_Vessel
  doneB   -> Controller.done

```

Für jeden Sender wird spezifiziert, für welche der Empfänger die einzelnen Outputpulse bestimmt sind, und welcher der Impulse jeweils getriggert wird. Diese Verknüpfung von Outputpulsen in Impulse stellt den "Glue" einer Interaktionsverbindung dar und wird als *Pulse Translation* bezeichnet.

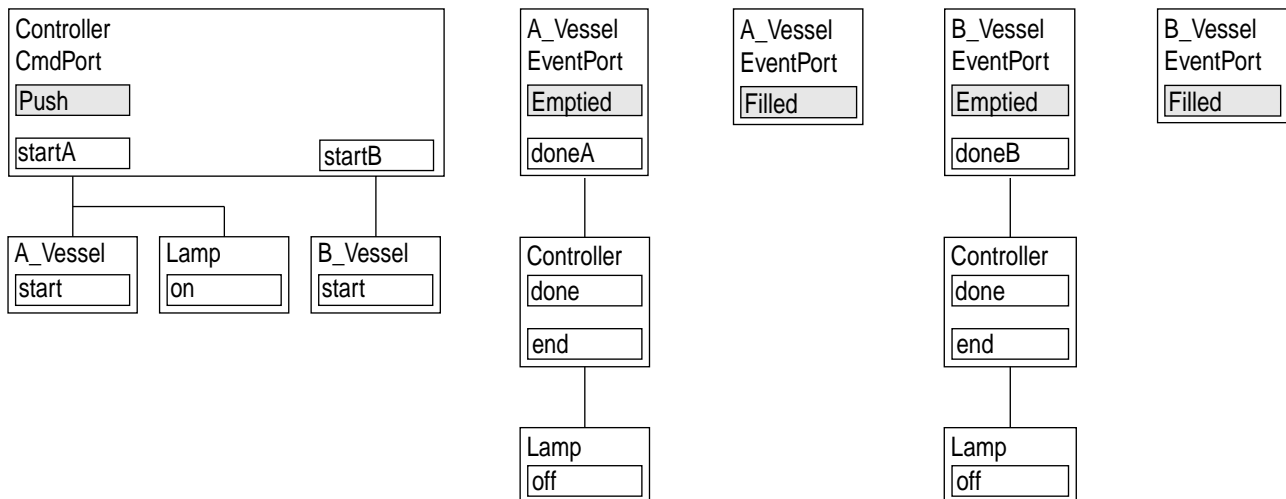
Prozessknoten mit mehreren möglichen Outputspulsen die Sequenzen alternativ verzweigen. Entscheidend am Interaction Analysis Tool ist, dass es zur Konstruktionszeit aktiv ist, das heisst die implizit definierten dynamischen Strukturen werden während der Entwicklung eines Modells sichtbar gemacht.

Das Werkzeug unterstützt damit den Anwender direkt beim Entwickeln der reaktiven System-Funktionalität. Die grafischen Interaktionsbäume machen die kausalen Abhängigkeiten im spezifizierten Verhaltensmodell deutlich und ermöglichen die Interaktion im Modell problemorientiert zu strukturieren. Dazu müssen normalerweise in iterativen Zyklen die Interaktion und zum Teil auch die Transitionstrukturen der Prozesse überarbeitet werden. Während der Überarbeitung kann ständig die Wirkung auf die Interaktionsbäume beobachtet werden.

Grosse Hilfe bietet das Tool zudem bei der Fehlersuche, bei Wartungsarbeiten, beim Kennenlernen bestehender Lösungen und bei Systemerweiterungen.

3.2. Interaktionssequenzen des Fallbeispiels *TwoVesselSystem*

Im Folgenden sind für das Fallbeispiel *TwoVesselSystem* alle Interaktionsbäume dargestellt, die eine externe Meldung als Ursprung haben. Jeder Baum zeigt die maximal möglichen Interaktionssequenzen, die durch die entsprechende Meldung bewirkt werden können. Die tatsächlich ausgeführte Sequenz hängt jeweils von den aktuellen Prozesszuständen ab. Eine angestossene Sequenz bricht ab, wenn ein Prozess keinen Outputspuls erzeugt, oder wenn ein getriggert Prozess den Inputspuls im aktuellen Zustand nicht erwartet.

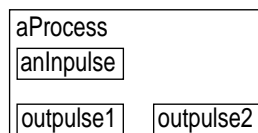
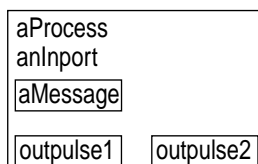


Die Meldung *Push* des Prozesses *Controller* kann den Outputspuls *startA* oder *startB* bewirken (siehe Transitionstruktur *PROCESS Controller*). Der Outputspuls *startA* triggert den Inputspuls *start* des Prozesses *A_Vessel* und den Inputspuls *on* des Prozesses *Lamp*, der Outputspuls *startB* nur den den Inputspuls *start* des Prozesses *B_Vessel*.

Die Interaktionsbäume für die Meldungen *Filled* von *A_Vessel* und *B_Vessel* bestehen lediglich aus dem Wurzelknoten, d. h. wenn diese Meldung auftritt, reagiert nur der entsprechende Prozess.

Die Meldung *Emptied* für *A_Vessel* oder *B_Vessel* bewirkt den Inputspuls *done* für den *Controller*, der abhängig von seinem Zustand mit dem Outputspuls *end* den Prozess *Lamp* zum Ausschalten der Lampe veranlasst.

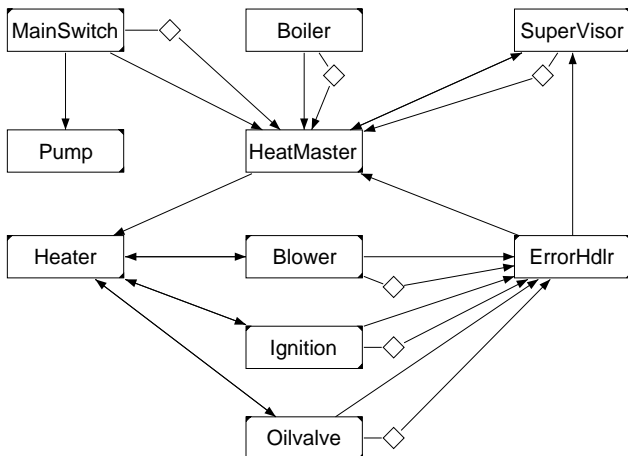
Legende:



3.3. Beispiel *HomeHeatingSystem*

Zur Illustration werden einige Interaktionsbäume eines komplexeren Clusters dargestellt. Es handelt sich um eine Heizungssteuerung, bei der die Steuerung von Gebläse, Zündung und Ölfluss mit Fehlerüberwachungen ausgestattet sind. Wenn ein Fehler auftritt, wird sofort das Ausschalten der Heizung eingeleitet.

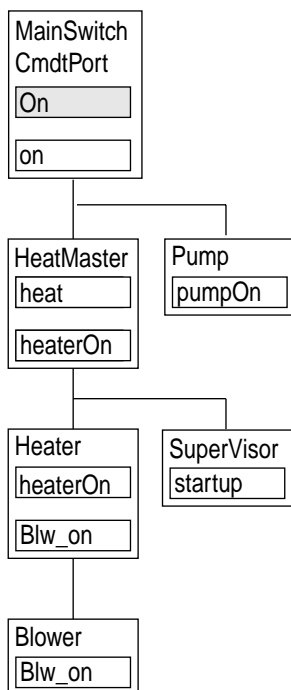
CLUSTER Heating INTERACTION NET



Die Netzstruktur zeigt, wie das Gebläse, die Zündung und der Ölfluss über den Prozess *Heater* durch die Prozesse *Blower*, *Ignition* und *Oilvalve* ein- und ausgeschaltet werden. Wenn einer dieser Prozesse einen Fehler detektiert, wird über den Prozess *ErrorHdlr* vom Prozess *HeatMaster* der allgemeine Abschaltvorgang eingeleitet, der auch eine Wirkung auf den Prozess mit der Fehlerdetektion haben kann. In diesen Fällen können Interaktionssequenzen auftreten, in welchen derselbe Prozess zweimal aktiviert wird.

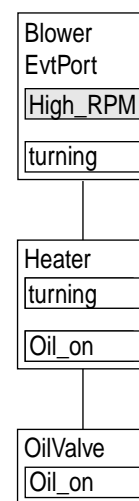
Einige Interaktionsbäume des CIP-Modells *HomeHeatingSystem*

Bei den folgenden Beispielen werden jeweils die maximalen Interaktionssequenzen erläutert. Abhängig von den aktuellen Prozesszuständen können Interaktionssequenzen früher abbrechen.



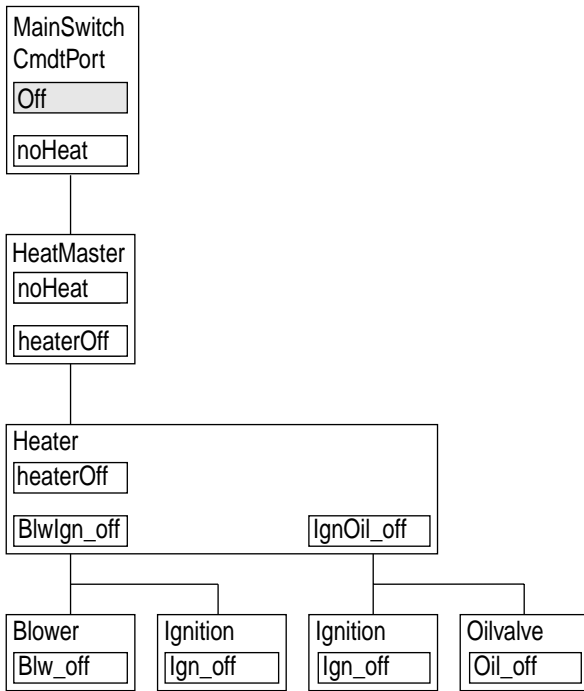
Das Einschalten des Hauptschalters führt über den Prozess *MainSwitch* zu einer Aktivierung der Prozesse *HeatMaster* und *Pump*.

Der *HeatMaster* startet beim *Heater* den Einschaltvorgang und informiert den *SuperVisor*. Der beim *Heater* getriggerte Einschaltvorgang führt dazu, dass der *Blower* das Gebläse startet.



Diese einfache Interaktionssequenz zeigt das Systemverhalten, wenn das Ereignis *High_RPM* (Drehzahl hoch) auftritt:

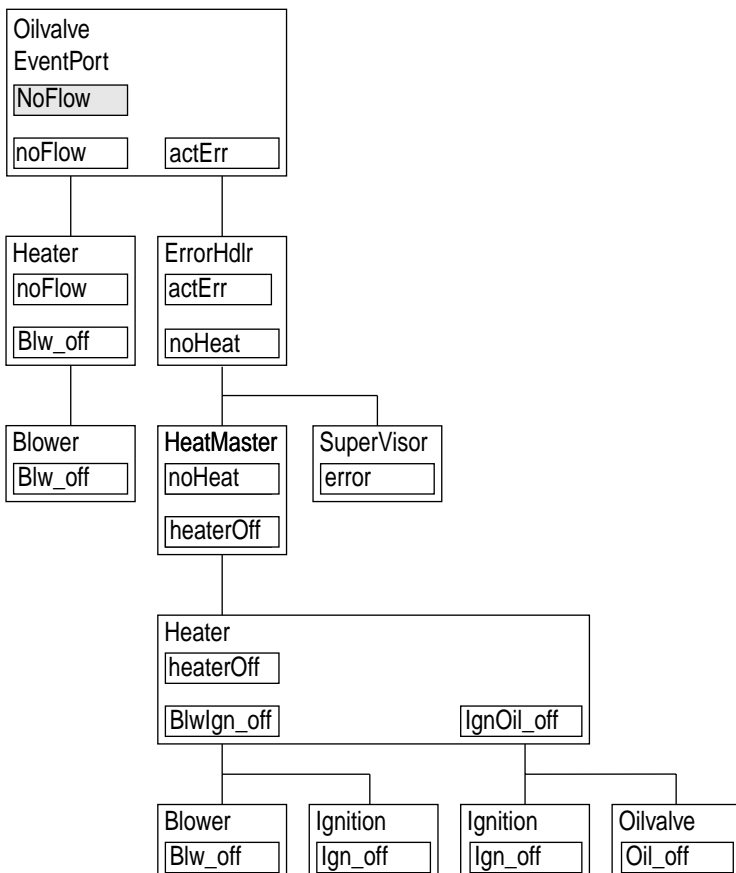
Auf den vom *Blower* erzeugten Impuls *turning* reagiert der *Heater* mit dem Output *Oil_on*, der als Impuls beim Prozess *Oilvalve* das Öffnen des Ölventils bewirkt.



Beim Ausschalten der Heizung hängt die Systemreaktion vom Zustand der Heizungssteuerung ab:

Auf den Input *heaterOff* reagiert der *Heater* entweder mit dem Output *BlwIgn_off* oder *IgnOil_off*.

Im ersten Fall wird das Gebläse und die Zündung deaktiviert, im zweiten Fall die Zündung und der Ölfluss.



Wenn kein Öl mehr fließt (Ereignis *NoFlow*), gibt es zwei mögliche Reaktionen:

Tritt das Ereignis *NoFlow* während des Abschaltvorganges der Heizung auf, reagiert *Oilvalve* mit dem Output *noFlow* für den *Heater*, was zum Ausschalten des Gebläses führt.

Andernfalls zeigt das Ereignis *NoFlow*, dass die Ölzufuhr nicht richtig funktioniert. Der Prozess *Oilvalve* reagiert in diesem Fall mit dem Output *actErr* für den Prozess *ErrorHdlr*, der sofort das Abschalten der Heizung einleitet.

Die vom *HeatMaster* bewirkte Pulspropagation ist gleich wie im vorhergehenden Beispiel, in dem das Ausschalten mit dem Hauptschalter bewirkt wird.

4. Komponentenbasierte Modellkonstruktion als CASE-Grundlage

CIP Tool ist als Komponenten-Framework aufgebaut. Die damit unterstützte komponentenbasierte Konstruktionstechnik, auch als "*White Box Composition*" bezeichnet, bildet eine wesentliche Voraussetzung für die automatisierte Interaktionsanalyse in CIP-Modellen.

Modellelemente wie Prozesse, Kanäle, Zustände oder Pulse sind eigenständige Komponenten, die beim Entwickeln erzeugt und konstruktiv miteinander verknüpft werden. Jede Komponente und jede Verknüpfung hat eine formale Bedeutung (Semantik), sobald sie vom Entwickler erzeugt worden ist. Verknüpfungen sind Assoziationen von Komponenten, die direkt im CIP-Datenmodell festgehalten werden. Operationen, Bedingungen und Funktionen werden zwar als C-Sprachkonstrukte definiert, gelten aber als *primitive* Komponenten, die mit den Transitionen erweiterter Zustandsmaschinen durch Zuordnung verknüpft werden. CIP Tool hat damit die Bedeutung eines Software-Modellbaukastens, bei dem jedes Modell in jedem Konstruktionszustand eine formale Semantik hat. Damit ist die Voraussetzung geschaffen, unvollständige Modelle automatisch zu analysieren und implizit definierte Modelleigenschaften darzustellen.

Eine weitere Voraussetzung für die automatische Interaktionsanalyse ist gegeben durch die problemorientierte Einschränkung des Entwurfsraums. CIP-Modelle bestehen aus statisch instanziierten endlichen Zustandsmaschinen, die über explizit spezifizierte Konnektoren interagieren. Die formale Basis für die automatische Analyse ist durch die Input/Output-Relationen der Zustandsmaschinen und die statisch definierten Übertragungsrelationen (Pulse Translation) der Interaktionskonnektoren gegeben. Die Beschreibung der Interaktionssequenzen abstrahiert von der Erweiterung der Zustandsmaschinen mit Programmierkonstrukten. Das ist möglich, weil diese Konstrukte als Primitive keinen Einfluss auf die Semantik der reinen CIP-Modelle haben.

5. Die Problematik sprachbasierter Modellspezifikation

Bei sprachbasierten Modellierungswerkzeugen sind automatische Analysefunktionen während der Modellierungstätigkeit nicht möglich, weil das eigentliche Systemmodell erst durch die Kompilation erzeugt wird. Formale Beziehungen zwischen verschiedenen Modell- oder Programmelementen entstehen erst, wenn der Parser in verschiedenen Ausdrücken identische Zeichenketten findet. Syntaktisch inkorrekte Spezifikationen und Programme können aus diesem Grund keine klare Semantik haben. Jeder Programmierer kennt diese Problematik aus Erfahrung. Bei Kompilationsfehlern zum Beispiel bezieht sich meistens ein erheblicher Teil der Fehlermeldungen auf Folgefehler, die nach dem Beheben der primären Fehler nicht mehr auftreten.

Die meisten in der Praxis bekannten grafischen Modellierungswerkzeuge sind zumindest teilweise sprachbasiert. Systemmodelle werden durch grafische Konstrukte spezifiziert, die mit syntaktischen Textkonstrukten beschriftet sind. Dabei können zwei Kategorien unterschieden werden:

a) Grafische Formalismen (Visual Formalisms)

Die bekanntesten Vertreter dieser Kategorie sind Statechartsmodelle [3]. Die Zustandsstruktur und die Zustandsübergänge werden durch grafische Modelle spezifiziert, was von Natur einen komponentenbasierten Charakter hat. Der Input und die Aktionen einer Transition werden hingegen durch spezifische syntaktische Textkonstrukte beschrieben. Die Interaktion zwischen orthogonalen Statechartskomponenten ist implizit durch die in den Textkonstrukten vorkommenden Ereignisnamen als verbindungsloser globaler Event-Broadcast definiert. Eine statische Interaktionsanalyse ist im Prinzip möglich, weil auch Statechartsmodelle in einem eingeschränkten Entwurfsraum definiert sind. Das Ausführen einer solchen Analyse während der Modellierung würde aber sehr schwerfällig, weil immer die syntaktische Korrektheit neuer und veränderter Konstrukte überprüft und das ganze Modell neu kompiliert werden muss.

Non-Deterministische und unbeschränkte Interaktionssequenzen werden von Statechartswerkzeugen denn auch erst zur Laufzeit detektiert. Das bedeutet aber, dass Sequenzen, die beim Testen nicht auftreten, nicht überprüft werden können.

b) Grafische Programmiermodelle

Zu dieser Kategorie zählen u. A. die neuen formalen Real-Time-Erweiterungen von UML [2, 8]. Architektur, Zustandsstruktur und die Zustandsübergänge werden auch hier durch grafische Modelle spezifiziert. Für die Beschriftung der Transitionen und die Definition der Interaktion zwischen Objekten werden aber Konstrukte einer objektorientierten Programmiersprache (C++) verwendet. Damit ist eine Überprüfung und Darstellung der Interaktionssequenzen erst zur Laufzeit möglich, das heisst nach Erzeugung und Kompilation der objektorientierten Programme. Die Modellanalyse müsste im Entwurfsraum einer objektorientierten Sprache stattfinden, was aufgrund der enormen Ausdruckskraft solcher Sprachen für reale Systeme aber kaum machbar ist.

Die Diskussion wurde hier auf Systeme mit statisch definierter Komposition beschränkt, d. h. während der Laufzeit können keine neuen Interaktionspfade und keine neuen Komponenten erzeugt werden. Bei objektorientierten Systemen mit Polymorphismus und dynamisch erzeugbaren Objekten wird die angeschnittene Problematik um Grössenordnungen schwieriger. Das ist auch der Grund, warum bei reaktiven und vor allem sicherheitskritischen Systemen der Entwurfsraum auf statisch komponierbare Modelle eingeschränkt werden sollte. Dynamisch komponierbare Modelle werden denn auch nach anderen Gesichtspunkten strukturiert, zum Beispiel als Client-Server Hierarchien mit Verantwortlichkeitsprotokollen für die von Servern zur Verfügung gestellten Diensten.

6. Der Nutzen der automatischen Interaktionsanalyse

Es ist naheliegend, dass die automatisierte Interaktionsanalyse kürzere Entwicklungszeiten und eine wesentliche Steigerung der Softwarequalität mit sich bringt, weil Modellierungsfehler vermehrt zur Konstruktionszeit entdeckt werden.

Ein Hauptproblem bei der Entwicklung reaktiver Echtzeitsysteme ist, eine problemorientierte Strukturierung der internen Systeminteraktion zu finden. Es ist offensichtlich, dass die explizite Darstellung der möglichen Interaktionssequenzen für die Lösung dieser Aufgabe von unschätzbarem Wert ist.

Weiterentwicklungen und Wartungsarbeiten setzen immer ein gründliches Erkennen und Verstehen der aktuellen Lösung voraus. Es ist klar, dass das die automatische Darstellung der Interaktionssequenzen bei diesem Einarbeitungsprozess enorme Vorteile bringt.

Literatur

1. Faergemand O. (ed.): SDL '93: Using Objects. Proceedings of the 6th SDL Forum. North-Holland (1993)
2. Selic, B., Gullekson, G. and Ward, P.T.: Real-Time Object-Oriented Modeling. (John Wiley & Sons, New York 1994). Tool: Rose Real-Time von Rational Software Corp.
3. Harel, D.: Statecharts: A Visual Formalism for Complex Systems. Science of Computer Programming, 1987, 8(3) pp. 231-274. Tool: Statemate von I-Logix Inc.
4. Rumbaugh J., Jacobson I., Booch G.: The Unified Modeling Language Reference Manual. Addison Wesley, 1999.
5. Fierz, H.: The CIP Method: Component- and Model-Based Construction of Embedded Systems. European Software Engineering Conference 1999 – ESEC 99. LNCS Vol. 1687, Springer Verlag Berlin (1999) 374-391.
6. Fierz H., H. Müller H.: Verhaltenssteuerung in reaktiven Systemen: Konstruktion von Master-Slave-Hierarchien in CIP Modellen. EKA' 99 - Entwicklung u. Betrieb komplexer Automatisierungssysteme, TU Braunschweig Mai 1999.
7. CIP Tool® 4.0, CIP System AG Zürich, 2000, <http://www.ciptool.ch>.
8. Harel, D., E. Gery: Executable Object Modeling with Statecharts. Computer (July 1997), 31-42. Tool: Rhapsody von I-Logix Inc.