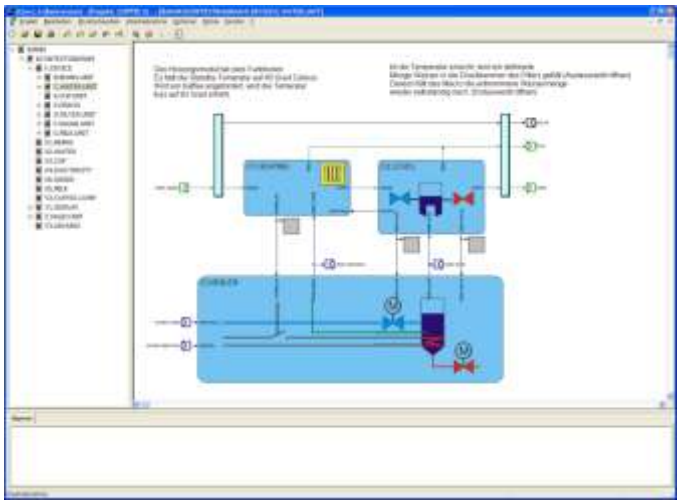
Modellgetriebene Softwareentwicklung in der Praxis oder wie man eine Kaffeemaschine systematisch programmiert.

Steffen Mertens ProSign GmbH
Januar 2008



„Erklärendes und planerisches Denken, insbesondere das ingenieurwissenschaftliche, ist weitgehend Denken in Modellen, oft mit starken Reduktionen. Der Mensch versteht die Welt offenbar nur über eine Reduktion der komplexen Realität auf wesentliche Eigenschaften in für ihn erfassbaren vereinfachten Darstellungen.“ * Heinz Duddek Akademie-Journal 1/2001

Die Verwendung von Modellen hat in der Softwareentwicklung eine lange Tradition, da Softwareprogramme zu den komplexesten Systemen gehören, die Menschen je geschaffen haben. Mit der Definition der Unified Modeling Language (UML) hat auch die Verbreitung von Softwaremodellen stark zugenommen. Allerdings werden Modelle häufig nur als gedankliche Verbindung zur realen Implementierung in der jeweiligen Programmiersprache verwendet. Sie sind also in der Regel „nur“ Dokumentation eines existierenden Softwaresystems oder Entwurfsgrundlage für die Softwarearchitektur. Eine solche Verwendung von Modellen nennt man *modellbasiert*. Die Modelle müssen ständig an den aktuellen Implementierungsstatus angepasst werden oder sie werden inkonsistent. Diese Art der Modellnutzung trägt nur mittelbar zur Weiterentwicklung der Programmiertechnologie bei. Nicht ganz zu unrecht werden solche Modelle von Softwareentwicklern als Overhead betrachtet. Teilweise wird auf Basis der Modelle Programmcode generiert, der dann für die folgenden Entwicklungsprozesse verwendet wird. Durch die Transformation auf Programmcodelevel wird allerdings eine Ebene erreicht, die für die Inbetriebnahme und Wartung eine zu hohe Detailtiefe besitzt und damit den Vorteil eines „echten“ Systemmodells verliert. Universelle Modellierungssprachen für Softwaresysteme wie z.B. die UML haben noch ein weiteres Problem. Durch die Konzentration auf die Softwareproblematik, wird die Implementierung meist nur anders dargestellt, ein echter Bezug zum Anwendungsgebiet (Domäne) ist nicht gegeben.

Die modellgetriebene Softwareentwicklung geht andere Wege und wird durch zwei wesentliche Konzepte geprägt. 1. Alle Änderung in der Software, werden ausschließlich im Modell vorgenommen. Die Transformation in die nächste Modellebene oder in das ausführbare Programm erfolgt automatisch. 2. Es werden domänespezifische Modellierungssprachen verwendet, die zum einen auf die Bedürfnisse der Fachexperten der jeweiligen Domäne zugeschnitten ist und zum anderen ist der Umfang der Sprachmittel auf das Anwendungsgebiet begrenzt. Insbesondere der zweite Punkt stellt mit der Abkehr von den universellen „standardisierten“ Modellierungssprachen einen Bruch zu den Ideen der UML dar und führt daher zu kontroversen Diskussionen.
Wenn man allerdings das komplexe, vielfach softwareunabhängige, technologische Know How betrachtet, das heute in Software umgesetzt wird, gibt es zu den domänespezifischen Modellen kaum eine sinnvolle Alternative. Die domänespezifischen Modellierungssprachen schließen die Lücken zwischen dem traditionellen Modelldenken der unterschiedlichen Ingenieursdisziplinen und der Implementierung in Software.
Ein Regelungstechniker denkt in Modellen, bei denen spezifische Funktionsblöcke den analogen Datenfluss darstellen. Ein Verfahrenstechniker kombiniert Rohrleitungen, Ventile und Behälter miteinander. In den domänespezifischen Modellen werden Objekte miteinander in Beziehung gesetzt, die dem Empfinden nach deutlich näher mit der realen Welt gekoppelt sind und gleichzeitig werden alle Aspekte ausgeblendet, die mit der unmittelbaren Problematik nichts zu tun haben.
Domänespezifische Modelle haben aber nicht nur Vorteile. Durch das Ausblenden domänefremder Aspekte fehlen sehr häufig Sprachmittel, mit denen eine strukturierte Problemanalyse des Gesamtsystems vorgenommen werden kann. So werden z.B. Ventile und Behälter in einer höheren Abstraktionsebene zu Einheiten kombiniert, für die andere Sprachmittel benötigt werden.
ProSign zeigt mit iCon-L einen Weg auf, wie man wesentliche Ziele der modellgetriebenen Softwareentwicklung praxisbewährt umsetzen kann.
Software-Engineering ist auch immer im Zusammenhang mit einem Vorgehensmodell für die Prozesse der Softwareentwicklung zu sehen. So wird nicht alleine durch die Verwendung eines Frameworks zur modellgetriebenen Softwareentwicklung eine gute Software entstehen. Gerade dann, wenn Technologen Software projektieren, ist es wichtig, eine Vorgehensweise aufzuzeigen, mit der gut strukturierte Lösungen entstehen.
In diesem Artikel soll anhand eines einfachen Beispiels dargestellt werden, wie mit dem Entwicklungswerkzeug iCon-L ein gut lesbares Softwaremodell entsteht, das automatisch in ein ausführbares Programm transformiert wird. Zudem wird dargestellt, wie durch eine integrierte Simulation ein Teil der Softwarevalidierung vor der Inbetriebnahme vorgenommen werden kann. Weiterhin zeigen wir, wie die Inbetriebnahme selbst und auch die spätere Wartung der Software effizient unterstützt werden kann.
In unserem Beispiel werden wir die integrierte HMI-Simulation aus dem DEMO-Paket von iCon-L programmieren. Die HMI-Simulation bildet das Verhalten einer Steuerung nach, welche mit einem kleinen Touchscreen 320*240 Pixel ausgestattet ist.

Abbildung 1: Intergierte HMI-Simulation

Die vorgestellte Software wurde für eine Schulung zur systematischen Anwendungsprogrammierung entwickelt und erhebt nicht den Anspruch der Vollständigkeit. Das Beispiel wurde so gewählt, dass für eine breite Anwendergruppe die Aufgabenstellung weitgehend klar umrissen werden kann. Zudem bietet diese Anwendung sowohl regelungstechnische als auch steuerungstechnische

Aspekte, die gerade in dieser Mischung für viele Maschinensteuerungen repräsentativ sind.

Beispiel ausprobieren

Betrachten wir zunächst das Beispiel sehr allgemein und schauen uns an, was das Programm eigentlich machen soll.

Wechseln Sie hierzu in den Betriebsmodus „Inbetriebnahme“ und wählen Sie die oberste Ebene (Main – Explorer-View).

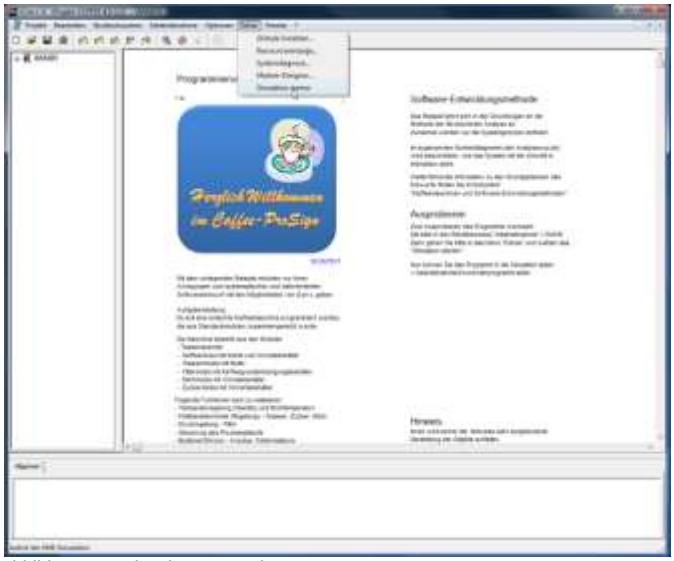


Abbildung 2: Main-Ebenen von iCon-L

Auf der obersten Ebene, der sogenannten Main-Ebene, ist in unserem Beispiel nur ein einziger „echter“ Baustein abgebildet. Weiterhin sind eine Reihe erklärender Textbausteine vorhanden. Textbausteine enthalten keine Funktionalität und werden bei der Erzeugung des ausführbaren Programms nicht berücksichtigt.
Die „echten“ Bausteine auf der Main-Ebene entsprechen Programm-Tasks. Eine Programm-Task wird entweder zeitzyklisch oder durch ein Ereignis aufgerufen. In unserem Beispiel wird der Programmbaustein zyklisch mit einer Zykluszeit von 50ms aufgerufen.

Gehen Sie nun eine Ebene tiefer, indem Sie einfach auf den Programmbaustein klicken. Sie befinden sich im sogenannten Kontextdiagramm. Abbildung 4

Rufen Sie nun im Menü *Extras* den Menüpunkt *Simulation starten* auf.

Auf Ihrem Bildschirm sollte nun ein Programmfenster mit dem Namen HMI-Simulation erscheinen.

Gehen Sie in das Menü *Inbetriebnahme* und rufen Sie den Menüpunkt *Anwenderprogramm laden…* auf. Beim ersten Aufruf werden sie aufgefordert, dass Zielsystem zu wählen. Wählen Sie den Eintrag > HMI-Simulation (Doppelklick oder Button *Verbinden*). Das Programmiersystem stellt nun eine Verbindung zur Simulation her. Aus Sicht des Programmiersystems ist die Simulation ein Zielsystem.
Nachdem Sie OK gedrückt haben, wird das Programm erzeugt. Sie erhalten eine Meldung über 56 Warnungen. Diese können Sie ignorieren, da diese Warnungen lediglich das Überlappen von grafischen Objekten auf dem Display anzeigen. Klicken Sie nun auf den Button *Laden*.

Abbildung 3: Aufwärmphase und Fehlermeldung

Das Programm simuliert zunächst die Aufwärmphase der Maschine und geht dann in eine Fehlerausgabe. Die Maschine zeigt an, dass Kaffee, Zucker und Milch fehlen.

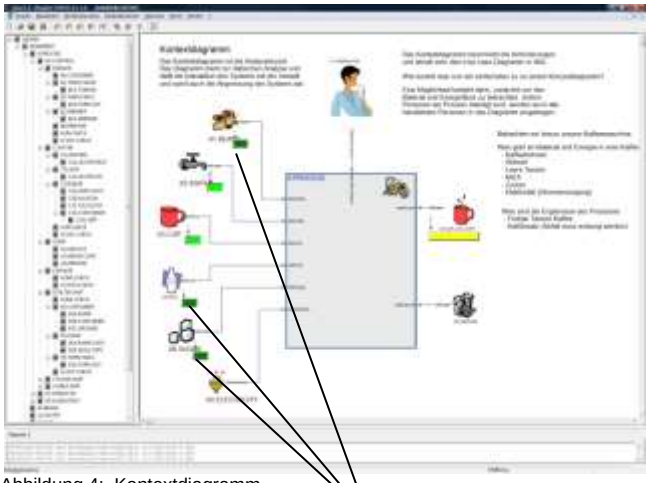


Abbildung 4: Kontextdiagramm

Klicken Sie auf die Button, um zu simulieren, dass Kaffee, Milch und Zucker nachgefüllt wurden

Nachdem simulierten Nachfüllen von Kaffee, Milch und Zucker sollten auch die Fehlermeldungen verschwinden und das Programm in den normalen Wartestatus für die Eingabe durch den Benutzer wechseln. Abbildung 5

Abbildung 5: Wartestatus für die Eingabe

Sie können nun mit einem Mausklick auf einen der Button einen Kaffee anfordern.

Abbildung 6: Bildfolge nach der Auswahl

Nachdem der Kaffee fertig ist, wartet das Programm auf das Entfernen der Tasse aus dem Ausgabebereich.

Klicken Sie für die Simulation der Tassenentnahme auf den gelben Button unter der vollen Tasse im Kontextdiagramm. Das Programm wechselt wieder in den Eingabestatus.

Schauen Sie sich nun an, wie das Programm im Fehlerfall reagiert. Indem Sie den Button X3.CUP drücken wird simuliert, dass keine neue Tasse in der Ausgabe vorhanden ist.

Abbildung 7: Fehlermeldung

Die Entwicklung der Software

Die Entwicklung der Software beginnt mit der Definition eines Kontextdiagramms, als Basis für die Anforderungsspezifikation. Das hier zur Anwendung kommende Verfahren lehnt sich an die Konzepte der strukturierten Analyse an, welche 1977 von Tom DeMarco entwickelt wurde. Das Kontextdiagramm ist die Wurzel des Analyse-Baums und stellt die Schnittstellen zur Umwelt dar. Bei dieser Anforderungsanalyse wird immer das Gesamtsystem in seinen definierten Grenzen betrachtet. In unserem Beispiel beschreiben wir dabei nicht das Steuergerät in seinen Grenzen, sondern den Kontext der Kaffeemaschine.
Dabei gehen wir zunächst von dem ganz allgemeinen Materialfluss der Maschine aus. Was geht in die Maschine rein? „Kaffeebohnen, Wasser, leere Tassen, Elektrizität, Milch und Zucker“ Was geht raus? „Kompletter Kaffe + Tasse und Kaffeesatz als Abfall“.
Je nach Betrachtung kann auch die Interaktion zum Anwender in das Kontextdiagramm eingebunden werden.

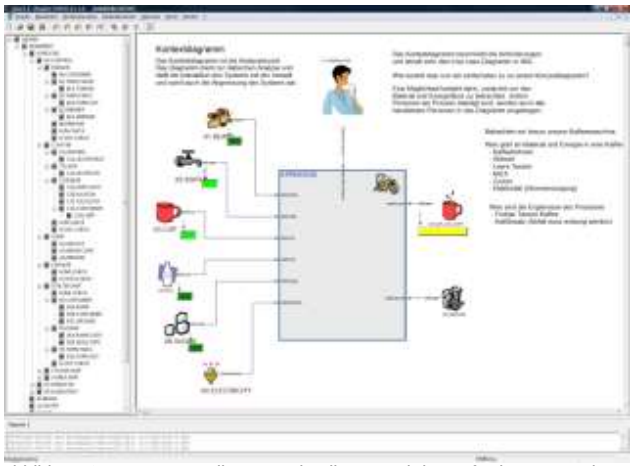


Abbildung 8: Das Kontextdiagramm ist die Wurzel der Anforderungsanalyse

Bei der Analyse sollten Sie so lange wie möglich versuchen, das Modell in einem Flussdiagramm zu beschreiben.

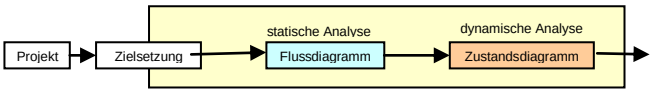


Abbildung 9: Systematisches Vorgehen in der Analysephase

Versuchen Sie zudem so lange wie möglich softwaretechnische oder systemspezifische Details auszublenden. Auch wenn Sie bereits wissen sollten, welche konkreten Sensoren Sie verwenden werden, sollten Sie diese Information im Kontextdiagramm oder auch in der nächsten Ebene noch nicht verwenden. Denken Sie immer an die Lesbarkeit Ihrer Modelle. Nicht jeder Betrachter des Modells kennt das reale Interface und weiß z.B. dass über einen speziellen Kontaktsensor das Vorhandsein einer leeren Tasse geprüft werden soll.
Auch in der nächsten Ebene, dem so genannten 0-Diagramm, wird versucht, soweit wie möglich die Regeln der statischen Analyse mittels Flussdiagramm einzuhalten.
Wie in Abbildung 10 dargestellt betrachten wir weiterhin den Materialfluss.

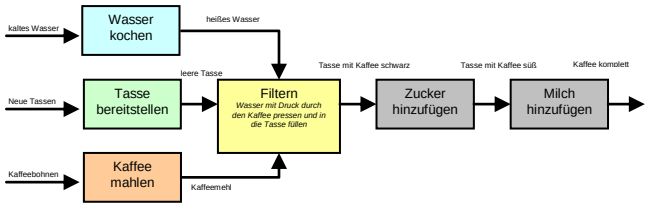


Abbildung 10 : Material und Energieflussanalyse im prozessorientierten Modell

Dabei wird der Materialfluss zunächst in einem prozessorientierten Modell betrachtet. Das 0-Diagramm soll in unserem Beispiel aber nicht nur den prozessorientierten Materialfluss darstellen, zusätzlich werden wir die Objektorientierung einführen.

Die Mischung von prozess- und objektorientierten Modellen ist sicher nicht ganz unumstritten, führt aber bei vielen Aufgabenstellungen zu effizienteren und besser lesbaren Lösungen. Zudem wird durch die Objektorientierung die Modularität der Software und damit sowohl Wiederverwendbarkeit als Wartbarkeit deutlich erhöht.

Soweit es möglich ist, sollten sich die Objekte an den realen Komponenten einer Maschine oder Anlage orientieren. In unserem Beispiel haben wir 5 eigenständige Komponenten.

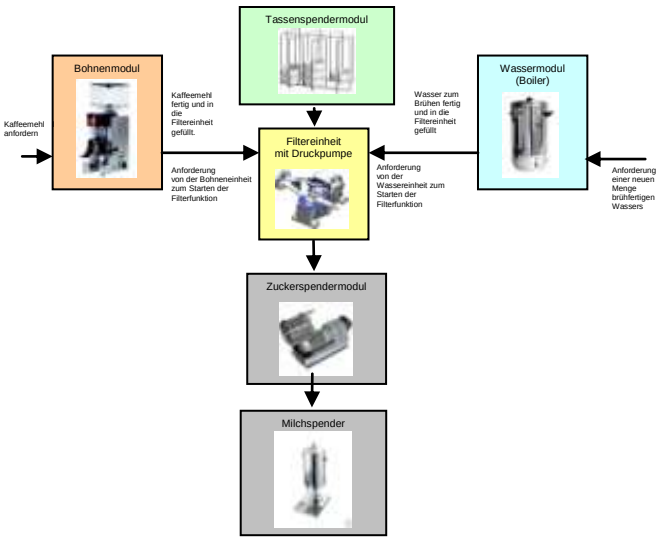


Abbildung 11: Vereinfachte Informationsflussanalyse im objektorientierten Modell

Wie bereits angedeutet, wird in unserem Beispiel das objektorientierte Modell mit dem prozessorientierten Modell kombiniert. Schaut man sich beide Strukturen an, erscheint dieses Vorgehen auch logisch, zumal bei den meisten Maschinen und Anlagen eine Zuordnung von Prozessen zu Objekten möglich ist. Neben den physisch vorhandenen Objekten wird ein zusätzliches Objekt für die Fehlerbehandlung eingeführt.

Da es sich bei den Objekten um eigenständig arbeitende Komponenten handelt, die auch Ihre Ressourcen (Material und Energie) eigenständig verwalten sollen, wird der grundlegende Materialfluss durch einen Informationsfluss ausgetauscht, bei dem lediglich die Information : Anforderung (Request) und Bestätigung (Confirm) als Information ausgetauscht werden. Im Grundsatz werden alle Objekte in einem ähnlichen Design implementiert.

Am Beispiel des „Wassermoduls“ soll dies verdeutlicht werden.
Das Wassermodul kümmert sich als Objekt selbstständig um alle Aspekte die mit dem Wasser zusammenhängen.

- Überwachung des Füllstandes (ständig)
- Regelung der Standby-Temperatur (ständig)
- Führungsregelung der Brühtemperatur (Anforderung)
- Bereitstellung der korrekten Wassermenge (Anforderung)

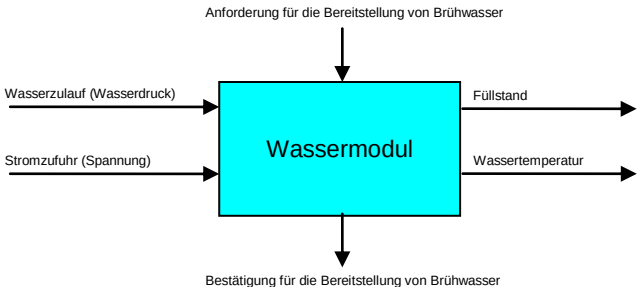


Abbildung 12: Objektorientierte Schnittstellenanalyse am Beispiel des Moduls zur Wasseraufbereitung

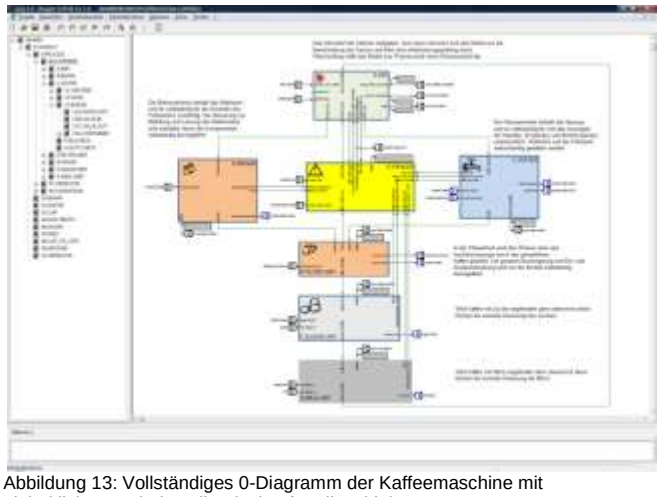


Abbildung 13: Vollständiges 0-Diagramm der Kaffeemaschine mit einheitlichem Schnittstellendesign für alle Objekte

In der Abbildung 13 können sie das vollständige 0-Diagramm der Kaffeemaschine in iCon-L sehen. Die Darstellung zeigt die einheitliche Umsetzung dieses Softwareentwurfs für alle Objekte. Mit dem gewählten Design wurden drei sehr wichtige Grundsätze für einen guten Entwurf erfüllt.

1. Die Anzahl der Schnittstellen zwischen den Objekte ist möglichst gering.
2. Die Objekte können weitgehend entkoppelt voneinander arbeiten und sind somit auch einzeln testbar.
3. Die Objekte kapseln sowohl die eigene Funktionalität als auch die selbst genutzten Ressourcen.

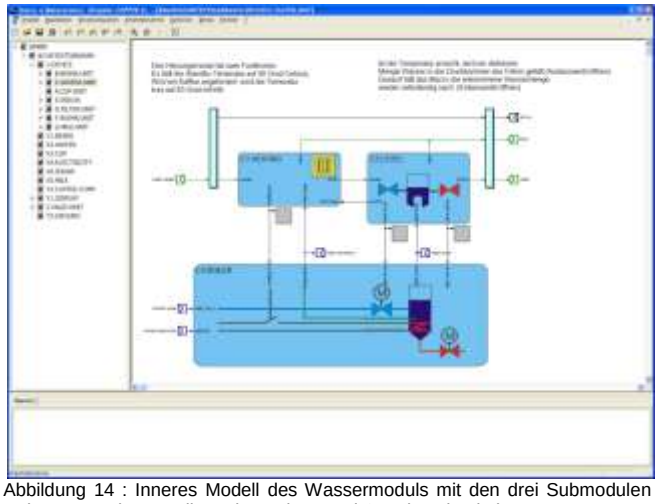


Abbildung 14 : Inneres Modell des Wassermoduls mit den drei Submodulen Heizungsregelung, Füllstandsregelung und Streckensimulation

Auch in den nächsttieferen Modellebenen wird, soweit möglich, versucht implementierungsspezifische Sachverhalte auszublenden. Wie zuvor soll dies am Beispiel des Moduls für die Wasseraufbereitung erläutert werden. Dieses Modul arbeitet als eigenständiges Objekt und gliedert sich in drei wesentliche Komponenten. Eine Einheit für die Regelung der Heizung, eine für die Regelung des Füllstandes und ein Modul repräsentiert den Boiler selbst. Das Boilermodul enthält eine Simulation mit der das Verhalten des Wasserbehälters mit der integrierten Heizung hinreichend genau nachgebildet werden soll.

Durch die Simulation soll es möglich sein, komplexe Prozesssituationen durchzuspielen und somit das Verhalten des Gesamtsystems zu testen. Außerdem kann die Simulation dazu genutzt werden, die Grenzleistungen einer Anlage zu ermitteln. Wie bereits im 0-Diagramm realisiert, wird der wesentliche Informationsfluss durch das Zusammenspiel der Anforderungs- und Bestätigungssignale geprägt. Lediglich für den Datenfluss zwischen den Reglermodulen und dem Boiler mit der Simulation werden echte Analogwerte ausgetauscht.

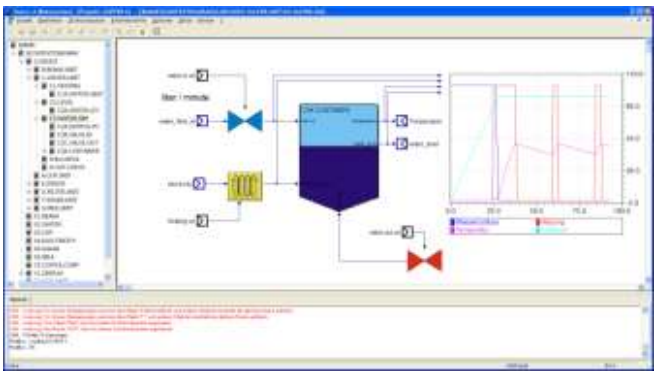


Abbildung 15 : Dynamisches Modell des Boilers in einer verfahrenstechnischen Modellierungssprache bei der Material- und Energiefluss betrachtet werden.

In der Abbildung 15 können Sie das verfahrenstechnische Modell für die dynamische Simulation des Boilers sehen. Das Modell ist so gestaltet, dass durch die schematische Darstellung der Anlagenkomponenten und deren Material- und Energiefluss ein weitgehend selbstdokumentierendes Modell entsteht. Die Objekte Ventil, Heizung und Behälter bauen sich wiederum aus kleineren Modellbausteinen auf. Um das interne Verhalten der Objekte nachzubilden wird ein weiteres Mal die Modellierungssprache gewechselt.

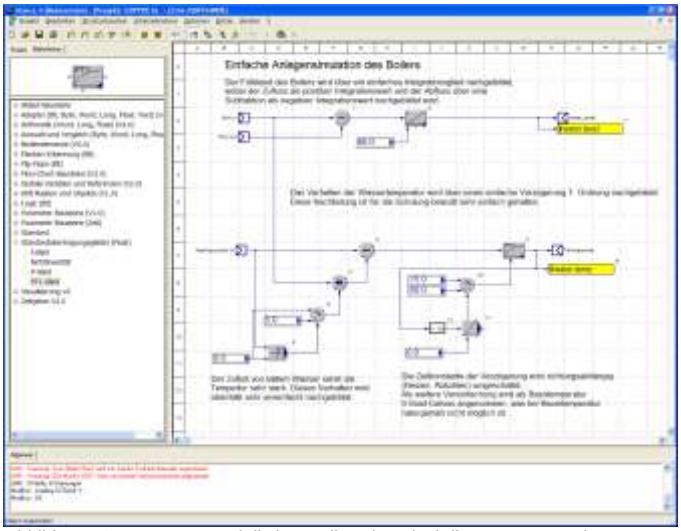


Abbildung 16 : Inneres Modell des Boilers im Blockdiagramm aus elementaren Übertragungsgliedern zur Modellierung dynamischer Systeme.

In der Abbildung 16 können Sie das vereinfachte mathematische Modell des Boilers sehen. Als Modellierungssprache wurde hier die klassische Darstellung im Blockdiagramm (Funktionsbausteinsprache) genutzt. Diese Modellierungssprache ist aus der Fachliteratur für Regelungstechnik hinlänglich bekannt und wurde in dieser Form auch schon vor der Verfügbarkeit leistungsfähiger Computersysteme zur Modellierung dynamischer Systeme verwendet. Da die Semantik der Bausteine durch unzählige Veröffentlichungen einem Quasistandard entspricht, ist das Verhalten der Modelle für viele Techniker und Ingenieure leicht nachvollziehbar.

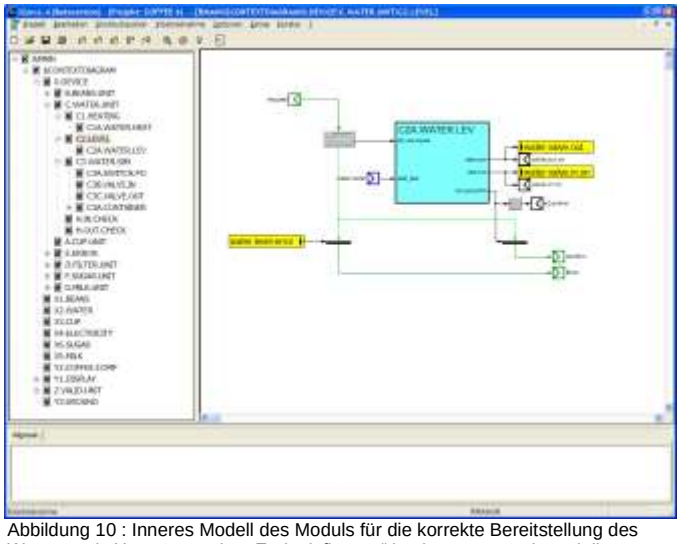


Abbildung 10 : Inneres Modell des Moduls für die korrekte Bereitstellung des Wassers als Umsetzung des „Ereignisflusses“ in einem Zustandsmodell.

Während die Modelle für die Simulation des Boilers den meisten Ingenieuren vertraut ist, sieht es bei der Umsetzung des Informationsflusses, welcher als Ersatz für den Materialfluss gewählt wurde, etwas anders aus. Im gewählten Softwaredesign kommt eine Besonderheit zum Tragen: Der Informationsfluss gehört eigentlich zur statischen Analyse, wird aber in dem hier vorgestellten Softwaredesign implizit zum Kontrollfluss. So liegt die Information, dass z.B. brühfertiges Wasser angefordert wird, nicht als „echtes“ Signal mit einem Wert an, sondern wird über ein „Bedingungs- und Ereignisnetzwerk“ nachgebildet. Wir haben es hier also wieder mit einem Wechsel der Modellierungssprache zu tun. Das verwendete Model entspricht den Konventionen von einfachen Petrinetzen, einem in der IT-Welt weit verbreitetem Modell für die Beschreibung von Zustandsmaschinen.

Bedingungs- und Ereignisnetzwerke basieren auf den Grundbausteinen Stellen und Transition, die über gerichtete Verbindungslinien (Kanten) verbunden werden.

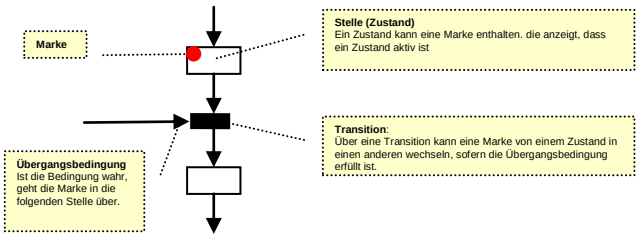


Abbildung 17 : Grundlagen eines Zustandsmodells als einfaches Bedingungs- und Ereignisnetzwerk

Zustandsmaschinen, die durch ein solches Bedingungs- und Ereignisnetzwerk modelliert werden, werden allgemein als recht sicher eingestuft, da ein Zustand immer nur über die definierte Kette von Stelle und Transition erreicht werden kann. Durch die grafische Darstellung der Zustandskette können Verzweigungen und Zusammenführungen sehr gut erfasst werden. Auch offene Ketten oder tote Zweige sind schnell erkennbar.

Übertragen wir nun das Verfahren des Ereignisflusses über eine Marke auf unser konkretes Beispiel. Solange wir uns noch nicht im inneren Modell des Reglers befinden, ist die Semantik der Signale sehr allgemein. Erst mit dem Eintritt in die Modellebenen der Regelungsmodule wird deutlich, dass die Verbindungslinien nur Marken übertragen können und keine „echten“ Signalleitungen mit Werten sind. In der Abbildung 18 sehen Sie, wie das Bedingungs- und Ereignisnetzwerk in das „Elternmodell“ integriert wird.

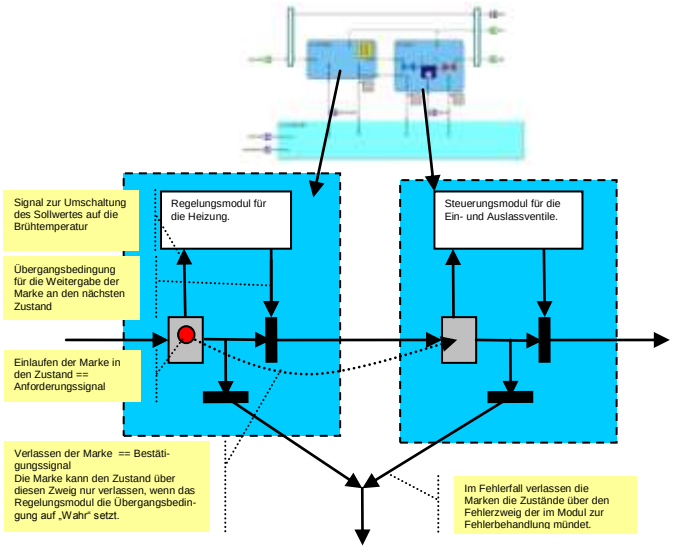


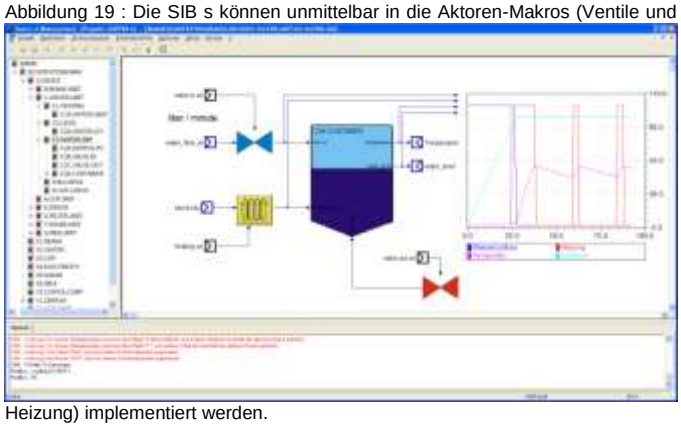
Abbildung 18: Kombination des Bedingungs- und Ereignisnetzwerkes mit den anderen Modellierungssprachen.

Die Aktivierung einer Stelle über eine Marke kann als logisches Signal von anderen Bausteinen abgegriffen werden. Dieses Signal wird genutzt, um z.B. ein angeschlossenes Regelungsmodul darüber zu informieren, dass das Wasser auf die korrekte Brühtemperatur zu bringen ist. Wurde diese Aufgabe von dem Modul erfolgreich erfüllt, dann wird die Bedingung der Transition auf „Wahr" gesetzt und die Marke an den nächsten Zustand weitergegeben. Ähnliches passiert auch im Fehlerfall, nur dass dann die Marke über einen anderen Zweig in eine andere Stelle überführt wird.

Aktoren und Sensoren

Wir wollen nun die reale Schnittstellenwelt der Steuerung, also die Aktoren und Sensoren, betrachten. Sensoren und Aktoren werden modellabhängig über kontextorientierte Serviceinterfacebausteine (SIB) repräsentiert. Betrachten wir hierzu die Abbildung 19. Ein Ventil und auch die Heizung werden direkt über den jeweiligen Baustein angesteuert. Unter diesen Bausteinen liegt dann der Zugriff auf die entsprechenden Services der Treiberschnittstelle zum realen E/A.

Abbildung 19 : Die SIB s können unmittelbar in die Aktoren-Makros (Ventile und



Heizung) implementiert werden.

Im Grunde wird das gleiche Konzept auch für die Programmierung des Touch-Screen-Displays unserer Kaffeemaschine verwendet. Die Schnittstelle zwischen Display und Anwendungsprogramm repräsentieren die genannten Serviceinterfacebausteine (SIB).

Abbildung 20 : Serviceinterfacebaustein für ein Displayobjekt von Typ Bitmap

Es gibt nur einen Unterschied. Während die E/A's in der Regel bereits existieren, müssen wir die grafischen Objekte im Display, die ja den Service erbringen sollen, erst noch schaffen. Daher wird die Programmierung des Displays in zwei formal getrennten Arbeitsschritten durchgeführt. Zunächst wird das vollständige Design der Bedienoberfläche gestaltet, wobei fertige grafische Grundobjekte (Bargraph, Button, Trend, formatierter Text usw.) zu komplexen Bedienseiten kombiniert werden. Hierfür können Sie leistungsfähige Hierarchiemechanismen nutzen, bei denen eine Kombination grafischer Objekte im Kontext von Masken dargestellt werden, wobei Masken wiederum Masken enthalten können. Im zweiten Schritt werden dann die Display-Objekete mit dem Anwenderprogramm verbunden, was über die SIBs realisiert wird. Sobald Sie einen SIB einfügen, wird der in Abbildung 21 dargestellte Designer aufgerufen.

Abbildung 21 : Designerwerkzeug zur Definition der Displayobjekte

Im Designer können Sie nun ein Objekt aus den bereits angelegten Objekten auswählen oder gegebenenfalls auch ein neues Objekt hinzufügen. Durch die Integration des Designer in das Framework entfallen die üblichen Aufwendungen für die Konfiguration der Schnittstellen, zudem ist die Synchronität der Darstellung zum treibenden Anwenderprogramm sichergestellt. Hierdurch können auch zeitkritische Darstellungen, wie z.B. Anzeigen zur Logikanalyse, realisiert werden.

Bisher haben wir nur die systematische Softwareentwicklung mittels domänespezifischer Modelle betrachtet. Nun wollen wir uns einem wesentlichen Vorteil der hier vorgestellten Technologie zur modellgetrieben Softwareentwicklung widmen. Aktuelle Erhebungen zu den Kosten von Software zeigen, dass ein großer Teil der Aufwendungen für Validierung, Test, Inbetriebnahme und Softwarepflege aufgebracht werden. Es soll nun gezeigt werden, wie das vorgestellte Entwicklungsprozessmodell helfen kann, diese Kosten erheblich zu senken.

Bei der Gestaltung der Modelle sollten Sie darauf achten, dass Sie immer ausreichend Beobachtungspunkte integrieren. Im hier vorgestellten Framework werden solche Beobachtungspunkte über Visualisierungsbausteine realisiert. In der Abbildung 19 können Sie sehen, wie ein Baustein zur Darstellung eines Trends in das Modell integriert wurde. Über diesen Trend kann sehr leicht überprüft werden, ob das erwartete Simulationsverhalten eingehalten wird. In der Abbildung 22 können Sie sehen, wie der aktive Zustand eines Moduls im Kontext des Prozessmodells angezeigt wird.

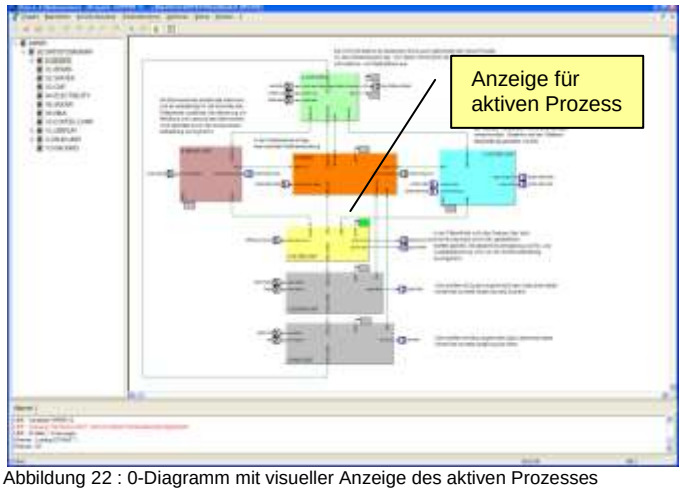


Abbildung 22 : 0-Diagramm mit visueller Anzeige des aktiven Prozesses

Um möglicht schnell einen Überblick zur Funktion des Gesamtsystems zu erhalten, wurde im vorliegenden Beispiel ein zusätzliches Validierungsmodul in die Software integriert. Über dieses Modul kann das gesamte System umfassend getestet werden. Zum einen werden alle wesentlichen Prozessströme und Ereignisse über Trendanzeigen visuell dargestellt, zum anderen können über dieses Modul definierte Fehlerzustände ausgelöst werden. Hierzu werden Sensoreneingänge mit fehlerhaften Werten überschrieben. Die in Abbildung 23 dargestellte Oberfläche zeigt die Bedien- und Visualisierungsebene des Validierungsmoduls.

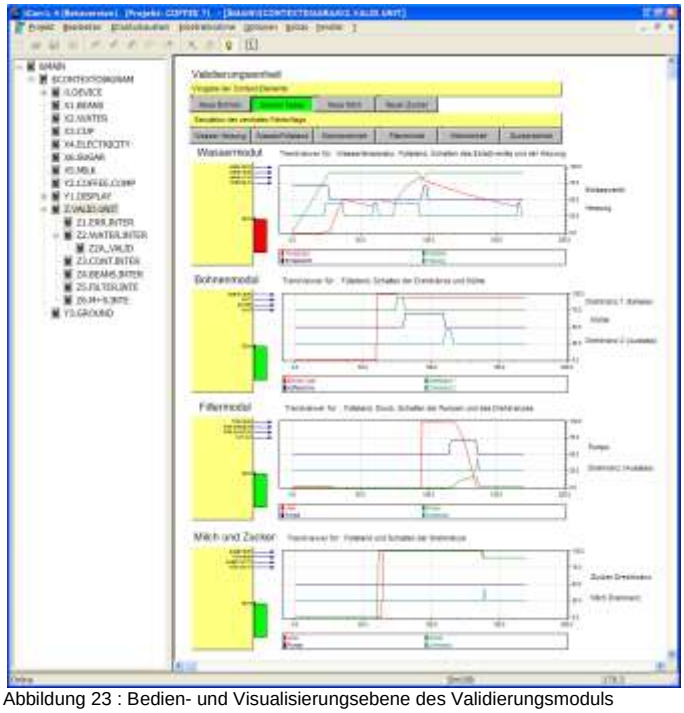


Abbildung 23 : Bedien- und Visualisierungsebene des Validierungsmoduls

Neben der visuellen Prüfung durch den Bediener, sollte ein Validierungsmodul auch eine permanente Überwachung kritischer Zustände enthalten. Diese Überwachung erfolgt vollständig unabhängig von der Realisierung der Anwendungssoftware und basiert auf vielen, sehr einfachen logischen Ausdrücken. Im Idealfall werden Validierungsmodule oder zumindest die Überwachungsmodule nicht von dem Mitarbeitern programmiert, der für die eigentliche Software zuständig ist. Für die Definition der Überwachungsmodule sollte der Programmierer nur die Aufgabenstellung kennen und daraus die nicht erwünschten Zustände ableiten.

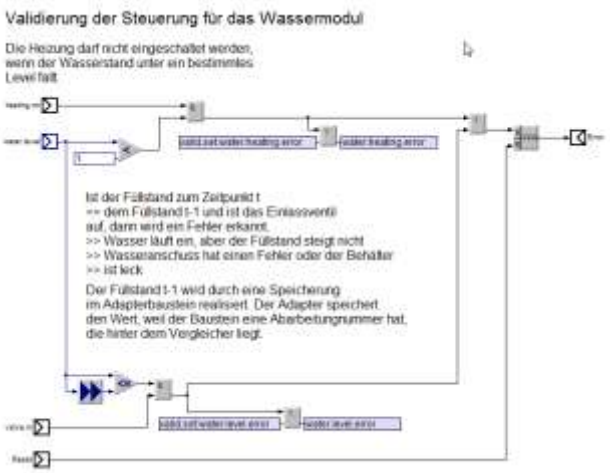


Abbildung 24: Beispiel für eine einfache Logik zur Überwachung der Wasserzufuhr und der Heizung.

Die Bedeutung der Validierung wird insbesondere dann erkennbar, wenn die Software bei der Inbetriebnahme geändert werden muss und für einen vollständigen Test keine Zeit mehr bleibt. Durch die integrierte Validierung wird ein Mindestmaß an Prüfung gewährleistet.

Die visuelle Funktionalität, welche für die Validierung im Simulationsmodus verfügbar ist, wird auch bei der eigentlichen Inbetriebnahme genutzt. Der einzige Unterschied besteht darin, dass anstatt der internen Simulation die reale Peripherie zugeschaltet wird. Da während des gesamten Entwicklungsprozesses nie die Modellsicht auf das Programm verlassen wird, werden dem Inbetriebnahmepersonal optimale Bedingungen zur Beobachtung der Software in die Hand gegeben. Im gewählten Beispiel sind zudem viele Modelle so nah an der realen technischen Auslegung des Systems angelehnt, dass eine zusätzliche Dokumentation vielfach nicht notwendig ist.

Fazit: Das vorgestellte Vorgehensmodell zur modellgetrieben Softwareentwicklung kann nach Ansicht des Autors einen wesentlichen Beitrag zur Entwicklung sicherer und pflegeleichter Embedded Software leisten. Durch das systematische Vorgehen mit dem Kontextdiagramm als Wurzel der Analyse und der Konzentration auf Material- und Energiefluss in den oberen Modellebenen wird ein Rahmen vorgegeben, der zielgerichtet zu einer gut dokumentierten Softwarelösung führt. Durch das systematische und ingenieurmäßige Vorgehen wird der Einfluss individueller Fähigkeiten des Programmierers geringer.