Inkrementeller Entwurf verteilter, eingebetteter Systeme mit VISTA

Stefan Eilers[2], Holger Krisp[2], Christian Müller-Schloer[2], Ralph Welge[1]

[1]**Welge GmbH**
Lüner Rennbahn 10
D-21339 Lüneburg
Internet: http://www.welge.de
e-Mail: ralph.welge@welge.de
Phone: +49-4131-32010 (Fax: -9990466)

[2]**Institut für Technische Informatik - Rechnerstrukturen und Betriebssysteme**
Universität Hannover
Appelstraße 4
D-30167 Hannover
e-Mail: {cms;eilers;krisp}@irb.uni-hannover.de
Phone: +49-511-762-19731 (Fax: -19733)

Zusammenfassung

Die wachsende Komplexität technischer Systeme und deren immer weiter gehende Vernetzung machen einen rechnergestützten Entwurf unabdingbar. Wir stellen mit VISTA einen Entwurfsprozess verteilter, eingebetteter Systeme vor, welcher unterschiedliche Konzepte (wie Systementwurf durch UML, Softwarebeschreibung und –generierung durch SDL, Multidomänenmodellierung und –simulation sowie Echtzeitsimulation) nahtlos miteinander kombiniert und damit einen inkrementellen Entwurf ermöglicht.

1 Einleitung

Die Komplexität technischer Systeme wächst weiterhin exponentiell. Dies wird besonders deutlich auf Chipebene, setzt sich aber fort auf der Ebene der Subsysteme und Supersysteme. Unter Subsystemen verstehen wir abgeschlossene Einheiten mit einem oder mehreren Prozessoren und zusätzlicher Hardware; Beispiele sind PCs, eingebettete Steuerungen, Mobiltelefone und PDAs (Personal Digital Assistants). Supersysteme entstehen durch Zusammenschaltung von Subsystemen über Kommunikationseinrichtungen. Künftige Kraftfahrzeuge entwickeln sich zu solchen Supersystemen, in noch größerem Maßstab sind aber darunter ganze Telekommunikationssysteme oder sogar das Internet zu verstehen. Ubiquitäre Systeme stellen einen Spezialfall dar, wobei die Unzuverlässigkeit der Verbindungen und die sporadische Verfügbarkeit von Ressourcen eine wesentliche Rolle spielen.

Derartige Systeme (oder Supersysteme) haben einige gemeinsame Charakteristika: Sie weisen neben der hohen Hardware-Komplexität immer größere Softwareanteile auf. Betriebssysteme und Middleware kommen verstärkt zum Einsatz. Neben rechnerartigen Subsystemen (Mikrocontroller, eingebettete Prozessoren zusammen mit ihrer Peripherie) sind weitere Komponenten für das Funktionieren des Gesamtsystems von Bedeutung: drahtgebundene oder insbesondere drahtlose Übertragungsmedien, elektrische, elektro-mechanische und mechanische Subsysteme bis hin zum menschlichen Bediener.

Die Rechnerunterstützung beim Entwurf von Systemen hat auf den untersten HW- und SW-Ebenen begonnen und sich kontinuierlich zu höheren Ebenen hin entwickelt. Die Ebene des Gesamtsystems wird dabei noch relativ wenig von Entwurfsautomatisierungs-(EA-) Werkzeugen unterstützt. Es existiert eine Vielzahl von einzelnen, domänen-spezifischen Werkzeugen, ihre Integration (und entsprechend die Integration der Subsysteme zum Gesamtsystem) ist aber noch nicht weit fortgeschritten. Dies liegt nicht nur an der Heterogenität der Domänen, sondern zusätzlich an der Vielzahl von Beschreibungssprachen, welche für jede der Domänen im Einsatz sind.

Ein Systementwurfsprozess (Abb. 1) sollte (mindestens) die folgenden Anforderungen erfüllen:

- Graphisch orientierte Modellierung des Gesamtsystems,
- verhaltensorientierte simulierbare Modellierung von Teilsystemen,
- Multi-language-Ansatz auf Subsystemebene (so, dass eingeführte domänen-spezifische Sprachen weiterverwendet werden können),
- besondere Unterstützung des SW-Entwurfs,
- Modellintegration, welche eine simulative Bewertung des Gesamtsystems erlaubt,
- wenigstens teilweiser Einsatz generativer Verfahren,
- Unterstützung des Entwurfs von Echtzeitsystemen,
- Unterstützung des Entwurfs von sicherheitskritischen Systemen.

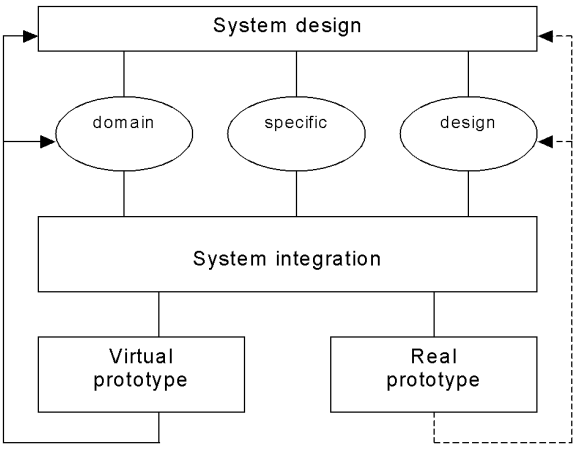

Abbildung 1: Systementwurfsprozess

Der vorliegende Artikel beschreibt das am IRB entwickelte Entwurfssystem VISTA (Virtual Prototyping of Safety- and Time-critical Systems), welches eine Reihe

dieser Anforderungen bereits erfüllt. Es unterstützt den Systementwurf mittels SDL oder ausgewählter Werkzeuge der UML-Methodik. Domänenspezifische Entwurfssprachen wie z.B. C, Modelica, EFSM oder Matlab/Simulink können benutzt und integriert werden. Für den SW-Entwurf mittels SDL (bzw. dessen Real-time-Erweiterung SDL.RT) existiert ein automatischer Codegenerator. Das Gesamtsystem kann hinsichtlich Funktion und Timing mit dem Systemsimulator Clearsim-MultiDomain bewertet und debugged werden.

Neben dem heutigen Stand der Entwicklung beschreibt der Artikel das Konzept (und erste Ergebnisse) der Methodik des inkrementellen Entwurfs. Ist ein System in Form eines virtuellen Prototyps beschrieben und simulativ bewertet, so muss es schrittweise realisiert werden. D.h., die Modelle einzelner Subsysteme werden durch die realen Komponenten ersetzt. Dieser verallgemeinerte Hardware-in-the-loop-Ansatz setzt jedoch eine exakte Synchronisation zwischen virtuellen und realen Systemteilen voraus. Da i.A. die realen Teile in Echtzeit laufen müssen, sind auch die virtuellen (also simulierten) Systemteile gezwungen, in Echtzeit oder schneller zu laufen. Geeignete Abstraktionen sowie der Einsatz von Echtzeitbetriebssystemen und schneller Hardware auf dem Simulationshost lassen erwarten, dass dieses Ziel in absehbarer Zeit erreicht werden kann.

Der vorliegende Artikel beschreibt die Methodiken und Werkzeuge für den Systementwurf mit UML (Kap. 2), den Softwareentwurf mit SDL (Kap. 3), die Systemsimulation (Kap.4) sowie den inkrementellen Entwurf (Kap.5).

2 Systementwurf mit UML

Mit ClearSim-MultiDomain ist es möglich, den Virtuellen Prototypen in schneller Geschwindigkeit und hoher Genauigkeit zeitlich und funktional zu validieren. Trotzdem wünscht sich der Entwickler auch eine bessere Unterstützung auf Systementwurfsebene, um den Beginn des Entwurfsprozesses übersichtlicher zu machen und so Fehler möglichst früh zu vermeiden.

Unser Ansatz verwendet mit UML eine Standardnotation, um den Entwurfsprozess mithilfe von VISTA geeignet zu erweitern.

2.1 Auswahl von geeigneten UML-Diagrammarten für den VISTA-Entwurfsprozess

Die Unified Modeling Language (UML) [1] ist eine Standardnotation, welche vorwiegend zur Beschreibung von objektorientierten Softwaresystemen verwendet wird. Sie besteht aus unterschiedlichen Diagrammen, wobei aber im Kontext des Entwurfs eingebetteter Systeme folgende Untermenge sinnvoll erscheint:

- **Use Case Diagramm**

Sog. Use Cases bzw. Anwendungsfälle bieten eine externe Sicht auf das eingebettete System und beschreiben die Primär- und Sekundärfunktionen, welche es der Umgebung anbieten soll. Diese Funktionen können hierbei entweder von anderen umgebenden Systemen oder auch menschlichen Benutzern verwendet werden. Die Bestimmung der Use Cases bildet den Startpunkt um alle Anforderungen des Systemes abzudecken (requirement engineering). Sie haben den großen Vorteil, dass sie für Entwickler und Kunde gleichermaßen verständlich sind und somit eine gemeinsame Diskussionsgrundlage über die gewünschte Systemfunktionalität bilden.

- **Klassendiagramm**

In objektorientierten Programmsystemen wird die Funktionalität durch miteinander kommunizierende Objekte hergestellt, welche Daten und Funktionen kapseln. Diese Objekte sind von Klassen abgeleitet, welche Gemeinsamkeiten zusammenfassen und das Konzept der Vererbung ermöglichen.

Analog hierzu sind auch eingebettete Systeme aus verschiedenen Komponenten aufgebaut, welche über genau festgelegte Schnittstellen und Protokolle miteinander kommunizieren. Im einfachsten Fall erfolgt der Datenaustausch über eine einzelne Digitalleitung (z.B. Verbindung zwischen einem CMOS-Baustein und einem Pin eines Mikrocontrollers) oder es liegt ein komplexeres Übertragungsprotokoll zugrunde (wie z.B. beim CAN Bus).

Der Vorteil der Aufteilung des zu entwerfenden Systems ist die Verringerung der Entwurfskomplexität, da nicht mehr ein großes, unübersichtliches Gesamtsystem entworfen und validiert werden muss. Stattdessen entwirft der Entwickler unterschiedliche Teilsysteme, welche über die spezifizierte Funktionalität und Schnittstellen verfügen, und setzt aus diesen getesteten Teilkomponenten das Gesamtsystem zusammen.

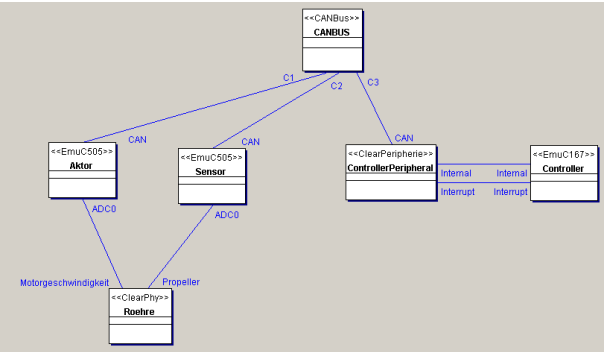

Abbildung 2 : UML Klassendiagramm

Für die Simulation ist wichtig, das sich auch Teile der Umgebung des eingebetteten Systems auf Klassen abbilden lassen. Diese Umgebungskomponenten beeinflussen mit ihren Eigenschaften maßgeblich das Verhalten des eingebetteten Systems (wie z.B. das Modell einer Windkanalröhre bei einer eingebetteten Regelung der Windgeschwindigkeit, s. Abb.2). Deshalb müssen alle für die Funktionalität des eingebetteten Systems notwendigen Umgebungsmodelle mit in das UML Klassendiagramm eingebunden werden.

Hat der Entwickler die Klassen des eingebetteten Systems und des Umgebungsmodells im UML Klassendiagramm festgelegt und durch Assoziationen die Kommunikationsbeziehungen spezifiziert, wird jeder Klasse eine domänenspezifische Beschreibungsform zugeordnet. Hierbei kann der Benutzer auf das gesamte Spektrum von ClearSim-MultiDomain angebotenen

Standarddomänen zurückgreifen (s. Kap. 4). Er kann sich so z.B. entscheiden, ein vorhandenes Servoventil über eine Kennlinie, Modelica oder EFSM zu modellieren. Die konkrete Wahl hängt vor allem von den Ansprüchen an die Genauigkeit des jeweiligen Modells und Aufwand für die Modellerstellung ab (u.U. könnte so z.B. auf ein schon erstelltes Modell in einer Bibliothek zurückgegriffen werden). Aufgrund der Kapselung der Klassen ist ein Wechsel zu einer anderen Modellierungsform einer Teilkomponente jederzeit problemlos möglich.

Aus diesem UML Klassendiagramm werden dann automatisch die für die Simulation nötigen Konfigurationsdateien erzeugt, so dass der virtuelle Prototyp validiert werden kann.

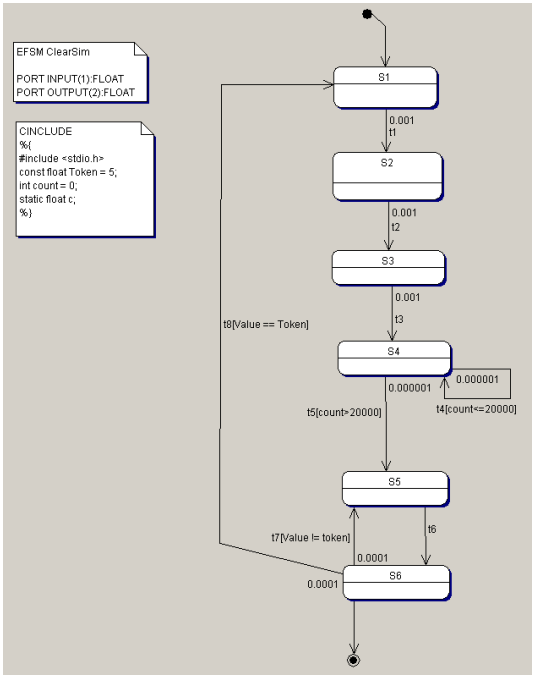

Abbildung 3: UML Zustandsdiagramm

- **Zustandsdiagramm**

Zustandsdiagramme in UML geben das Durchlaufen einzelner interner Zustände eines Objektes abhängig von den von anderen Objekten empfangenen Nachrichten wieder. Sie zeigen damit das dynamische funktionale Verhalten eines Objektes zur Laufzeit an. Da UML ursprünglich für die Notation von objektorientierten Softwaresystemen entwickelt wurde, fehlt bis heute ein einheitlicher semantischer Unterbau der Zustandsdiagramme, so daß wir für den Entwurf eingebetteter Systeme eine eigene Semantik definiert haben.

Das von UML angebotene Zustandsdiagramm (verwandt mit den weit verbreiteten Statecharts [2]) soll dann auf zwei Arten verwendet werden können:

1. Das Verhalten eines Simulationsmodules wird anstelle einer EFSM durch ein UML Zustandsdiagramm ausdrückt. Hierbei werden die vom Benutzer im Zustandsdiagramm spezifizierten Eingaben (Zustände, Transitionen, Übergangsbedingungen, Timing und ANSI C Code Annotation) automatisch in die textuelle EFSM-Beschreibung umgesetzt und in ein Simulationsmodul für ClearSim-MultiDomain umgewandelt. In der Simulation wird dann die durch das UML Zustandsdiagramm dargestellte Domäne durch die Funktion und das Timing des spezifizierten endlichen Automaten modelliert.
2. Die Eingaben des Zustandsdiagrammes werden direkt an einen Codegenerator weitergegeben, welcher daraus ein ablauffähiges Programm erzeugt. Dieses Laufzeitsystem (in ANSI C implementiert) kann dann für die Zielplattform compiliert (und mit unserem Simulator ClearSim-MultiDomain simuliert) werden. Insbesondere ist ein Vergleich des realen Timings mit dessen Spezifikation (s. Punkt 1) möglich.

- **Sequenzdiagramm**

Sequenzdiagramme zeigen die Interaktion von verschiedenen Objekten zur Laufzeit an. Bei eingebetteten Systemen können diese Diagramme zur Beschreibung des Kommunikationsverhaltens der unterschiedlichen Teilkomponenten verwendet werden. Insbesondere können Zeitschranken (timing constraints) spezifiziert werden, welche wegen Anforderungen an die Echtzeitfähigkeit unbedingt eingehalten werden müssen. In Zusammenhang mit ClearSim-MultiDomain planen wir, diese Zeitschranken in der Simulation zu validieren und die UML Sequenzdiagramme mit den erhaltenen Simulationsergebnissen zu annotieren.

2.2 Der Design-Prozess mit UML und VISTA

Den Ausgangspunkt des Designprozesses mit UML bilden *Use Case Diagramme* des zu entwerfenden eingebetteten Systems, in welche die Informationen der Anforderungsanalyse des Entwicklers einfliessen. Diese Diagramme geben wieder, welche Grundfunktionalitäten das System anbieten muss und werden aufgrund des Pflichtenheftes erstellt. Vorteilhaft in dieser Phase ist auch die Rücksprache mit dem Kunden, da er aufgrund der leicht verständlichen Form der Use Cases direkt seine Wünsche und Forderungen wiederfinden kann.

Hat man durch die Use Cases eine semi-formale Spezifikation des Systems erstellt, so wird im zweiten Schritt vom Entwickler die Struktur im UML *Klassendiagramm* festgehalten, welches auch die Kommunikationsbeziehungen der einzelnen Systemteile untereinander festlegt. Hierbei trifft der Entwickler z.B. Entscheidungen darüber, welche Teile durch Hardware und welche durch Software realisiert werden sollen. Zu den Bestandteilen, welche zum eigentlich zu entwerfenden System zugehören legt er auch die Komponenten der Umgebung fest, welche für die Simulation der Systemfunktionalität von Bedeutung sind.

Komponenten, welche anfänglich nur mit einer groben Verhaltenspezifikation beschrieben und simuliert werden sollen, werden mit UML Zustandsdiagrammen beschrieben. In diesen wird die Funktionalität über ANSI C Code und zusätzlichen Zeitangaben festgelegt und sie werden automatisch in ein Simulationsmodul umgewandelt. Über den Codegenerator kann aus den Zustandsdiagrammen auch ein ablauffähiges Programm erzeugt werden: hierbei wird aber das vorher spezifizierte Timing durch das Ablaufverhalten auf dem realen

Mikrocontroller ersetzt, welches dann die Simulation präzise vorhersagt.
Im Laufe des Entwicklungsprozesses werden alle virtuellen Komponenten durch reale Implementierungen ersetzt, so dass der Virtuelle Prototyp langsam zu einem Realen Prototyp wird. Auf diesem Weg kann auch frühzeitig die reale Umgebung eingebunden werden. Dieser Ablauf wird ausführlicher in Kapitel 5 beschrieben.

3 SDL.RT-Softwareentwurf mit VISTA

VISTA enthält eine integrierte SDL.RT-Entwicklungsumgebung für den funktionalen und zeitorientierten Entwurf sowie die Implementierung eingebetteter zeit- und sicherheitskritischer Systeme.

3.1 Die Basissprache ITU-SDL

Die SDL.RT-Entwicklungsumgebung (s. Abb. 4) basiert auf der grafisch repräsentierten, durch die ITU standardisierten Sprache SDL. Der ursprüngliche Schwerpunkt von ITU-SDL für kommunizierende Systeme kommt der heutigen, oftmals durch das Internet geprägten Forderung nach Interoperabilität stark entgegen. Mit ITU-SDL werden Systeme auf der Basis einer idealisierten Maschine modelliert. Auf diese Weise können komplexe Mechanismen einfach formuliert werden. Ein System ist damit das Modell einer Software und nicht die Software selbst.
Die wesentlichen Charakteristiken von ITU-SDL sind folgende:

Struktur: Es bestehen Ausdrucksmöglichkeiten zur formalen Notation einer hierarchischen Dekomposition. Damit kann ein komplexes System übersichtlich beschrieben werden.

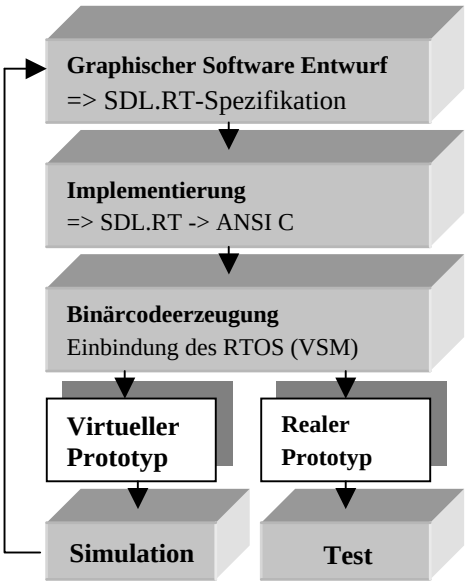

Abbildung 4: SDL-basierter Entwurfsablauf

Verhalten: Aktive Softwareelemente werden als Prozesse modelliert. Prozesse sind kommunizierende erweiterte, endliche Automaten (CEFSM = Communicating Extended Finite State Machine). Prozesse bestehen aus Zuständen und Transitionen, die nach dem RTC-Konzept (Run To Completion) ausgeführt werden. Die Semantik der Sprache schreibt vor, dass die idealisierte Maschine beliebig viele dieser Prozesse parallel ausführen kann. Über die Nebenläufigkeit der aktiven Softwareelemente hinaus besteht aus Sicht der Prozesse eine Ortstransparenz. Physikalische Grenzen sind daher nicht Bestandteil eines Modells.

Kommunikation: ITU-SDL schreibt eine nachrichtenorientierte Kommunikation zwischen Prozessen vor. Über die Verwendung von Anweisungen zum Empfangen und Versenden von Daten sowie Kanälen und Signalrouten kann die Modellierung verteilter, verständlicher und robuster Softwaresysteme vorgenommen werden.

Zeit: Neben einer Systemzeit wird ein Timer-Mechanismus zur dezentralen, an Prozessgrenzen gebundenen zeitlichen Selbstbeaufschlagung angeboten.

Daten: Für die Modellierung von Datenstrukturen wird das Konzept der *Abstrakten Datentypen* (ADT) angeboten.

3.2 Die Entwurfssprache SDL.RT

Das Profil von ITU-SDL kann mit dem rein funktionalen Entwurf von reaktiver Software für nachrichtenorientierte Echtzeitsysteme beschrieben werden. Zur Unterstützung des hier geforderten Profils der eingebetteten zeit- und sicherheitskritischen Systeme wurde die Sprache ITU-SDL zur Sprache SDL.RT [3] hin erweitert.

Die Grundüberlegungen bei der Entwicklung von SDL.RT waren folgende: ITU-SDL soll vollständig in SDL.RT enthalten sein. Die SDL.RT-Erweiterungen sollen mit den Basiskonzepten von ITU-SDL verträglich sein. Schließlich sollen die Erweiterungen nur dort vorgenommen werden, wo das erweiterte Profil es erfordert. Es wurden drei neue Sprachgruppen eingeführt:

SDL.RT-Zusicherungen: Mit SDL.RT-Zusicherungen wird entwurfsseitig eine fokussierte Möglichkeit geschaffen, logische, datenbasierte Ausdrücke über den Zweck einer oder mehrerer Transitionen einzugeben. Zur Laufzeit werden diese Ausdrücke bewertet. Sind die Ergebnisse logisch „wahr“ so werden die Ausführungen der jeweiligen Transitionen als korrekt bezeichnet. Es besteht damit eine Möglichkeit zur Beobachtung des Softwareverhaltens. Mit einer *SDL.RT-precon-Anweisung* kann der Start einer Transition abgesichert werden. Eine *SDL.RT-postcon-Anweisung* überprüft die Prozessintegrität am Ende eine Transition. Eine *SDL.RT-prccon-Anweisung* kann prozessweit mit den jeweiligen *SDL.RT-postcon-Anweisungen* verknüpft werden. Für den Fall des Fehlverhaltens einer Software kann eine *SDL.RT-exception-Prozedur* spezifiziert werden. In dieser Prozedur wird die Prozessintegrität wiederhergestellt und optional mittels einer *SDL.RT-repeat-Anweisung* ein Wiederaufsetzverfahren eingeleitet.

SDL.RT-Shared-Memory-Signalisierung: Die SDL.RT-Shared-Memory-Signalisierung ergänzt die von ITU-SDL übernommene nachrichtenorientierte Signalisierung. Mit dem neuen Kommunikationsmecha-

nismus wird die Modellierung eng gekoppelter Prozesse möglich. Wesentliche Aspekte sind eine statische sowie eine dynamische Deklaration, eine nachrichtenorientierte Zugriffsbenachrichtigung und die Einführung wechselseitigen Ausschlusses sowie Synchronisation über CEFSM-verträgliche Semaphoren.

SDL.RT-Ereignisplanung: Die SDL.RT-Ereignisplanung kann als zentrales Instrument zur Modellierung des funktionalen und zeitorientierten Systemverhaltens eingesetzt werden.

3.3 Ebenen des Softwareentwurfs

SDL.RT ist eine Entwurfssprache zur funktionalen und zeitorientierten Entwicklung von Softwaremodellen. Ein Entwurfsvorgang beginnt in der Regel auf einer abstrakten Ebene mit der Entwicklung eines Strukturmodells, das in weiteren Schritten um funktionale und zeitorientierte Eigenschaften ergänzt wird. Ein komfortables Hilfsmittel zur Erstellung von Strukturmodellen ist die *Hierarchische Dekomposition*. Die *Hierarchische Dekomposition* stellt mehrere Diagrammtypen zur Verfügung.

Systemspezifikation: Die Systemspezifikation ist die höchste Ebene eines Systementwurfs. Hier wird die statische Deklaration der Blöcke und deren Interaktion vorgenommen. Für die Modellierung der Kommunikation stehen Kanäle zur Verfügung, die zeit- und wertdiskrete Signale, d.h. Nachrichten, transportieren. Signale können in *SDL.RT-text-Anweisungen* deklariert und in *SDL.RT-signallist-Anweisungen* den Kanälen zugeordnet werden. Die idealisierte SDL.RT-Maschine übernimmt den Vorgang des Transfers.

Blockspezifikation: Blockspezifikationen werden der mittleren Hierarchiestufe zugeordnet. Eine Blockspezifikation ist die Redefinition eines Blockes durch eine Blocksubstruktur in einer tieferen Hierarchieebene. Für die Blockmodellierung stehen die grafischen Anweisungen der Systemspezifikation zur Verfügung.

Prozessinteraktionsdiagramm: Die jeweils unterste Blockspezifikation eines jeden Blocks enthält aktive Elemente, die sogenannten Prozesse. Alle Prozesse eines Systems arbeiten parallel. Die über Signale kommunizierenden Prozesse können durch Services, d.h. durch weitere CEFSMs, unterstrukturiert werden.

Prozessspezifikation: Prozesse oder deren Services enthalten einen Prozessgraphen und einen Prozessdatenbereich. Entsprechend dem CEFSM-Konzept besteht ein Prozessgraph aus Zuständen und Transitionen. Komfortable grafische Anweisungen zum Empfangen und Versenden von Signalen sowie eine auf gemeinsamem Speicher basierende Signalisierung erleichtern die Interprozesskommunikation. Prozeduralen Elementen wie Prozeduren und Makros ergänzen die Möglichkeiten.

3.4 Implementierung

Eine integrierte Implementierungsmethodik übersetzt SDL.RT-Modelle in prozeduralen C-Code, der dann mit handelsüblichen C-Compilern in die sogenannte Systemimplementierung, d.h. in Binärcode, überführt werden kann.

Wesentliche Bestandteile der Systemimplementierung sind die Prozessimplementierungen, das nachrichtenorientierte Laufzeitsystem VSM (Virtual SDL.RT-Machine) und ein Mikrokern. Die VSM emuliert die Eigenschaften der idealen Maschine und gewährleistet mit einer Prozess-, einer Signal- und einer Geräteverwaltung die Nebenläufigkeit der Prozesse sowie die nachrichtenorientierten Systemcharakteristiken. Der Mikrokern stellt im wesentlichen eine Speicherverwaltung und eine Abstraktion der realen Hardwareebene zur Verfügung. Eine weitere derzeit noch nicht voll realisierte Aufgabe des Mikrokerns besteht in der Gewährleistung der Ortstransparenz. Während Anwendungen wie die Internettechnik zeigen, dass eine kommunikationsseitige Ortstransparenz realisierbar ist, ist die Ortstransparenz für verteilten gemeinsamen Speicher wegen eines hohen Komplexitätsgrades nicht vertretbar.

3.5 Simulation

Mit dem Simulator ClearSim-MultiDomain kann die Systemimplementierung auf der Basis eines virtuellen Prototypen ausgeführt werden. Funktionale und zeitorientierte Verhaltenseigenschaften werden in das Modell der ausgeführten Software zurückgeschrieben. Mit Sequenzdiagrammen kann die Systemimplementierung bewertet werden.

Einen genaueren Überblick über die Möglichkeiten der System-Simulation vermittelt der folgende Abschnitt.

4 System-Simulation mit ClearSim-MultiDomain

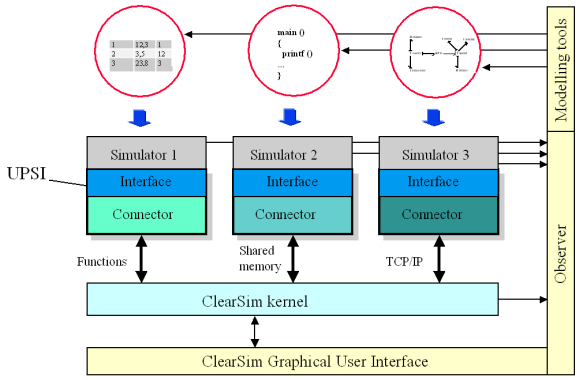

Abb. 5 : Aufbau von ClearSim-MultiDomain

Hat der Entwickler das Programm für den Mikrocontroller entworfen (z.B. graphisch unterstützt mit SDL.RT, s. Kap. 3), so ist dessen korrekte Funktion im Gesamtsystem zu validieren. Die korrekte Funktion wird bei eingebetteten Systemen aber nicht nur durch den logischen Ablauf des Steuerprogrammes festgelegt: es müssen fast immer auch bestimmte Funktionen in einer Mindestzeit abgearbeitet werden, damit das Gesamtsystem korrekt arbeitet (Echtzeitkriterium).

Will der Entwickler die funktionale und zeitliche Korrektheit seines Entwurfes nachweisen, so kommt aber

erschwerend hinzu, das der Mikrocontroller eng mit seiner heterogenen Umgebung (mechanische und elektronische Komponenten) gekoppelt ist. Eine Möglichkeit wäre das Einsetzen der Komponenten in die reale Umgebung und die Ausführung von konkreten Tests, welches aber zumeist kostspielig oder manchmal sogar unmöglich ist (z.B. weil die umgebende Hardware noch nicht fertiggestellt ist).

Ein besserer Ansatz ist die Verwendung eines Systemsimulators, welcher in der Lage ist, das funktionale und zeitliche Verhalten des *gesamten* eingebetteten Systems schon zu einem frühen Entwicklungszeitpunkt im Computer zu bestimmen. Anhand des sog. Virtuellen Prototypen, welcher als Simulationsmodell im Rechner existiert, können dann schon früh Aussagen über die Richtigkeit des Entwurfes getroffen werden, welches Entwicklungszeit und -kosten reduzieren hilft.

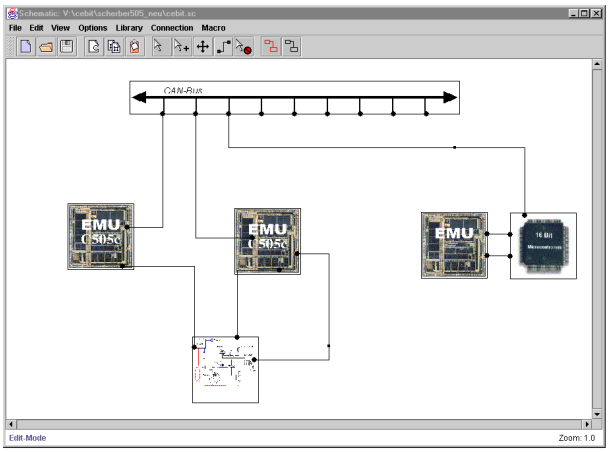

Abbildung 6: Java-Oberfläche für Konfiguration der Simulation

Ein wesentlicher Punkt beim Systemsimulator ist die Art der Modellierung der einzelnen Systemkomponenten. Hierbei gibt es Ansätze, welche versuchen das Gesamtsystem in einer einheitlichen Beschreibungssprache zu modellieren [4, 5, 6], aber eine befriedigende Lösung ist auf diese Art unserer Ansicht nach nicht zu erreichen. Einerseits müsste diese Sprache sehr umfangreich (und dadurch unübersichtlich) sein, um alle nötigen Konzepte für angemessene Modellierung der unterschiedlichen Domänen zu enthalten. Andererseits müssten die Entwickler erst diese neue, komplexe Sprache lernen, um mit ihr vernünftig arbeiten zu können.

Vielversprechender scheint uns das Konzept der heterogenen Modellierung und Simulation zu sein, auf welchem der von uns entwickelte ereignisbasierte Systemsimulator ClearSim-MultiDomain basiert (s.Abb.1). Hierbei werden an einen gemeinsamen Simulationskernel über eine offene Schnittstelle (UPSI[1]) unterschiedliche Simulationsmodule angekoppelt, welche dann jeweils für die Modellierung und Simulation einer Teilkomponente des Systems verantwortlich sind. Der Simulationskernel ist hierbei für den Austausch von Ereignissen (d.h. Daten mit angehängtem Zeitstempel) zwischen den Modulen zuständig und sorgt für das Simulationscheduling, in dem er den Einzelmodulen die Berechtigung zur Simulation und die zu simulierende Zeitscheibe vorgibt.

Der Benutzer kann hierbei die unterschiedlichen Systemkomponenten auf verschiedene Arten beschreiben:

- Ausführbares Programm auf Mikrocontroller Infineon C167 und C505c inklusive Nachbildung aller integrierten Peripheriebausteine
- Extended Finite State Machines (mit ANSI C)
- Can Bus[2] Kommunikation
- Modelica Beschreibung / PDGL's
- SDL.RT Beschreibungen
- Matlab/Simulink Modell

Zusätzlich zu diesen standardmäßig vorhandenen Beschreibungsarten, welche alle auf Knopfdruck direkt in Simulationsmodule für ClearSim-MultiDomain umgesetzt werden können, ist durch die offene Schnittstelle zum Simulationkernel auch die Einbindung von kommerziellen Tools problemlos möglich. So wurden z.B. die kommerziellen Simulatoren Matlab/Simulink und Siemens Plantsim sowie das Visualisierungswerkzeug Siemens WinCC erfolgreich an ClearSim-MultiDomain angebunden.

Hat der Benutzer die einzelnen Simulationsmodule erstellt, so kann er mittels einer Java-Oberfläche die Simulationsstruktur graphisch festegen. Hierbei stellt er die Parameter der einzelnen Module ein (z.B. nötige Eingabedateien) und legt die Verbindungen fest, über die Module Daten miteinander austauschen können (s.Abb. 6).

Danach kann über die Oberfläche die Simulation gestartet werden. Die Simulationsperformance hängt hierbei von der Komplexität der einzelnen Simulationsmodule und der Anzahl der ausgetauschten Datenereignisse ab. Der in Abb. 6 gezeigte Aufbau[3] benötigt zur Simulation nicht einmal die doppelte Zeit auf einem gängigen PC (Pentium mit 450MHz Taktfrequenz, WinNT 4.0) verglichen mit der zu simulierenden Realzeit. Ersetzt man den C167 (16 bit) durch einen C505c (8 bit) mit ähnlichem Steuerprogramm, so wird sogar nur noch ein Viertel der zu simulierenden Zeit zur Berechnung der Simulation benötigt.

Diese Simulationsperformance macht den Simulator sehr gut für den interaktiven Entwicklungsbetrieb einsetzbar. Bei denen dem Simulator beigelegten Simulationsmodulen ist die Genauigkeit der zeitliche Vorhersagen typischerweise besser als 1%, die Funktionalität wird exakt nachgebildet.

5 Inkrementeller Entwurf

Bei dem Design komplexer eingebetteter Systeme ist die ausschließlich virtuelle Überprüfung der funktionalen sowie zeitlichen Korrektheit eines Prototypen nicht befriedigend, da der sprunghafte Übergang zum realen Prototypen noch viele Probleme aufwerfen kann, die aufgrund der statischen virtuellen Umgebung nicht

[1] Unified Portable Simulation Interface [7]

[2] Feldbussystem aus dem Automobilbereich

[3] Steuerung mit drei über CAN Bus gekoppelten Mikrocontrollern (ein Infineon C167 und zwei C505c) an einem Windkanal (mit Modelica modelliert)

sichtbar werden. Müssen Systeme mit ihrer Umwelt kommunizieren, so sind es natürlich gerade die nicht vorhersehbaren Ereignisse die zur Fehlfunktion des Prototypen führen. Dies gilt besonders bei offenen Systemen, da hier nicht gewährleistbar ist, daß die möglichen Kommunikationspartner sich der Erwartung gemäß verhalten. In diesem Fall steckt der Entwickler in dem Dilemma, daß bei optimaler Beobachtbarkeit des virtuellen Systems keine Probleme auftreten, da sich die virtuelle Umgebung ja stets innerhalb der vorgegebenen Randbedingungen bewegt, aber der schlecht beobachtbare reale Prototyp in seiner realen Umgebung Fehlverhalten zeigt. Die notwendige Fehlersuche treibt nun die Kosten der Entwicklung in die Höhe.

Damit diese Probleme möglichst zu einem frühen Zeitpunkt der Entwicklung auftreten, bei dem noch eine gute Beobachtbarkeit des Systems existiert und somit eine kostengünstige Fehlersuche möglich ist, sollte ein inkrementeller Entwurfsansatz mittels Echtzeitsimulation gewählt werden:

Wurde der virtuelle Prototyp erfolgreich in seiner virtuellen Umgebung überprüft, so muß er sich gegenüber der realen Umgebung beweisen. Dazu werden diejenigen Simulationskomponenten entfernt, welche vorher die virtuellen Kommunikationspartner repräsentierten und durch Schnittstellenkomponenten ersetzt, welche nun mit der realen Umgebung verbunden werden.

Es wird also eine sogenannte »Mixed Reality« Simulation durchgeführt, unter Einbeziehung einer realen Umgebung. Treten nun Fehler auf, so sind sie somit mittels der guten Beobachtbarkeit virtueller Systeme schnell und kostengünstig auffindbar und behebbar.

Ist diese Phase der Verifikation abgeschlossen, wird der noch existierende virtuelle Prototyp Element für Element aus der virtuellen Umgebung entfernt und als reale Komponente an den Simulator angeschlossen. Dabei wird zweckmäßigerweise mit den Elementen begonnen, die als unkritisch angesehen werden. Diejenigen die hohen Softwareanteil besitzen (also z.B. Mikrocontroller), werden als letztes aus der Simulation entfernt da die virtuellen Beobachtungs- und Debuggingmöglichkeiten hier besonders wichtig sind.

Diese Vorgehensweise garantiert einen Entwicklungsprozess welcher größere Entwicklungssprünge verhindert und somit die Kosten für ein eventuell notwendiges Redesign minimiert.

Die sogenannte »Mixed Reality« Simulation verlangt neben garantiertem Echtzeitverhalten eine hohe Simulationsgeschwindigkeit, da permanent die Synchronität der virtuellen Simulationszeit mit der Realzeit garantiert sein muß. Bisher war dies nur mittels aufwendiger und teurer Spezialhardware möglich, welche schnell veraltete und somit nur eine geringe Investitionssicherheit garantierte.

Da aber heute auch normale Workstations oder PC's aufgrund der inzwischen hohen Rechenleistung immer komplexere (und damit realitätsnahere) Simulationen in Echtzeit durchführen können, wurde der Schwerpunkt auf die Verwendung handelsüblicher Computer gelegt. Dies hat den Vorteil der Weiternutzung der Simulationsrechner für traditionelle Aufgaben, wenn die Geschwindigkeitsanforderungen für Simulationen nicht mehr ausreichend zur Verfügung gestellt werden kann.

Um die Echtzeitsimulation zu erforschen wurde ein komplett neuer Simulationskernel für ClearSim-MD entwickelt, welcher auf einem timergesteuerten Algorithmus basiert. Zusammen mit der Möglichkeit Mehrprozessorrechner auf Shared-Memory Basis zu verwenden, konnte so die Realisierbarkeit einer Hochgeschwindigkeitssimulation mit deterministischem Zeitverhalten demonstriert werden. In diesem Zusammenhang wurde ebenfalls die Mixed-Reality Simulation erfolgreich praktisch überprüft, indem ein virtueller Can-Bus über eine Schnittstelle in der Simulation an einen realen Can-Bus angeschlossen wurde. Über diesen Bus regelte ein virtueller Mikrocontroller (Infineon C505) einen Modell-Windkanal, indem er mit realen Mikrocontrollern (Sensor für die Windgeschwindigkeit und Motorsteuerung für den Propeller) kommunizierte, die ebenfalls an den Can-Bus angeschlossen wurden (s. Abb. 7). Diese Technik der Schnittstellen zwischen virtueller und realer Welt soll in Zukunft auch auf digitale und analoge Signale erweitert werden.

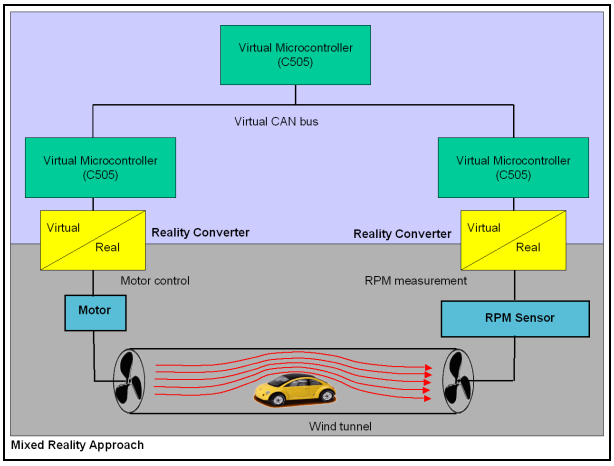

Abbildung 7 Echtzeitsimulation am Beispiel Windkanalsteuerung

Beim Vergleich mit einem realen C505 Mikrocontroller, welcher exakt die gleiche Regelungssoftware abarbeitete, war eine gute Vergleichbarkeit zwischen dem Mixed-Reality und dem komplett realen System nachweisbar. Somit konnte der inkrementelle Entwurfsansatz vom komplett virtuellen Prototypen über die Mixed-Reality Simulation bis zum komplett realen Prototypen erfolgreich demonstriert werden.

Nicht verschwiegen werden soll die Tatsache, daß die Simulation kompletter Mikrocontroller in Echtzeit noch immer davon abhängt wie viel Prozent Idle-Anteil, bei dem der Prozessor auf Interrupts wartet, bei der Abarbeitung eines Programms besteht. Hierbei kann bei einem 450 Mhz Pentium III Prozessor ein Slowdown von 2 bei vollständiger Belastung (0% Idle) gemessen werden. Es ist aber zu erwarten, daß sich dieser Umstand aufgrund der ständig anwachsenden Prozessorleistung stetig verbessern wird.

6 Zusammenfassung und Ausblick

Wir haben in diesem Artikel einen idealen Systementwurfsprozess und seine teilweise Realisierung mit den Werkzeugen der VISTA-Umgebung beschrieben. Die Werkzeuge SID und Clearsim-MultiDomain befinden sich im praktischen Einsatz und sind als Produkt[4] verfügbar. Neben dem im Artikel beschriebenen Windkanal wurden z.B. ein ABS-System, eine Walzensteuerung sowie ein Diagnose- und Managementsystem für ein Telefonortsnetz modelliert. Die UML-Methodik befindet sich in der Erprobung, ebenso die Echtzeitsimulation [8].

Ein Projekt, welches die VISTA-Methodik speziell in Richtung auf den Entwurf sicherheitskritischer Systeme erweitert, befindet sich in der Anfangsphase. Ziel ist es hier, logische und temporale Zusicherungen (assertions) in den System- und/oder Domänenentwurf einzubringen und sowohl simulativ wie auch real (durch ein generiertes Diagnosesystem) zur Laufzeit zu überwachen. Zur Benutzerinteraktion werden die UML-Sequenzdiagramme verwendet. Formal basiert die Assertionspezifikation auf der Design-by-contract-Methode von B. Meyer und auf der Sprache ADL.

References

[1] James Rumbaugh et al: The Unified Modeling Language Reference Manual, Addison-Wesley 1998

[2] David Harel, Michal Politi: Modeling Reactive Systems With Statecharts : The Statemate Approach, McGraw Hill 1998

[3] Welge, R.: SDL.RT basierter Entwurf und Implementierung eingebetteter, zeit- und sicherheitskritischer Systeme, Dissertation an der Universität Hannover, 2001

[4] A. A. Jerraya, K. O`Brien: SOLAR: An Intermediate Format for System-Level Modeling and Synthesis, article in Codesign: Computer-Aided Software/Hardware Engineering, IEEE Press, 1995

[5] J. Zhu, D. Gajski: OpenJ: An Extensible System Level Design Language, Proceedings of the Design, Automation and Test in Europe Conference and Exhibition 1999

[6] D. L. Barton: The Structure of High Level Systems Descriptions, Proceedings of the Forum on Design Languages, Lausanne, 1999

[7] Scherber,S.: Modellierung und Simulation softwareintensiver eingebetteter Systeme, Dissertation an der Universität Hannover, 2001

[8] Holger Krisp, Jochen Bruns, Stefan Eilers, Christian Müller-Schloer: Multi-Domain Simulation for the Incremental Design of Heterogeneous Systems, ESM 2001, Prag, Juni 2000

[4] Die VISTA-Tools werden durch die Welge GmbH, Lüneburg, kommerziell vertrieben.