Informatik Forschung und Entwicklung manuscript No.
(will be inserted by the editor)

AutoFOCUS – Ein Werkzeugprototyp zur Entwicklung eingebetteter Systeme*

Manfred Broy, Franz Huber, Bernhard Schätz

Institut für Informatik, Technische Universität München, Arcisstr. 21, D-80333 München
(e-mail: {broy, huberf, schaetz}@in.tum.de)

Eingang: Juli 1999 / Annahme:

Zusammenfassung. Der Beitrag stellt AutoFOCUS vor, einen Werkzeugprototyp zur Entwicklung verteilter, eingebetteter Systeme auf der Grundlage formaler Techniken. AutoFOCUS unterstützt die Systementwicklung mit integrierten, im wesentlichen graphischen Beschreibungstechniken, mit deren Hilfe sowohl unterschiedliche Sichten als auch verschiedene Abstraktionsebenen eines Systems beschrieben werden. Um konsistente und vollständige Beschreibungen sicherzustellen, bietet AutoFOCUS die Möglichkeit, Konsistenzbedingungen zu formulieren und Systembeschreibungen daraufhin zu überprüfen. Aus ausführbaren Spezifikationen können Prototypen des entwickelten Systems erzeugt werden und in einer Simulationsumgebung ausgeführt und visualisiert werden. Zur formalen Verifikation von Systemeigenschaften verfügt AutoFOCUS über Anbindungen an Modellprüfungswerkzeuge wie μ-cke oder SMV.

Schlüsselwörter: Software Engineering Werkzeuge, Entwicklung verteilter und eingebetteter Systeme, formale Methoden und Techniken, Prototyping

Abstract. This article presents AutoFOCUS, a tool prototype for the development of distributed, embedded systems based on formal techniques. AutoFOCUS supports system development offering integrated, comprehensive and mainly graphical description techniques to specify different views as well as different levels of abstraction of a system. To avoid ill-defined specifications, consistency conditions on these system descriptions can be formulated and checked. Prototypes can be generated from executable specifications using a Java code generator. These prototypes can be executed and visualized within a simulation environment. System Properties can be formally verified using model checking tools such as μ-cke or SMV.

* Die in diesem Beitrag vorgestellten Arbeiten wurden im Rahmen des DFG-Sonderforschungsbereichs 342 (“Werkzeuge und Methoden für die Nutzung paralleler Rechnerarchitekturen”), des Projektes “SYSLAB” durch das Leibniz-Programm der DFG und durch Siemens-Nixdorf, sowie des Bayerischen Forschungsverbunds Software Engineering FORSOFT gefördert.

Key words: software engineering tools, development of distributed and embedded systems, formal methods and techniques, prototyping

CR Subject Classification: D.2.2, D.2.4, D.3.1, F.3.1, I.6.4

1 Einführung

Eingebettete Systeme haben in den letzten Jahren in vielen Anwendungsbereichen massiv an Bedeutung gewonnen, angefangen in klassischen Aufgaben der Steuerungs- und Regelungstechnik bis hin zu multimedialen Anwendungen in der Unterhaltungselektronik. Unter einem eingebetteten System versteht man dabei im allgemeinen ein System aus einer Kombination von Hard- und Softwarekomponenten, die in ein technisches System integriert sind und dieses System steuern und überwachen.

Zunehmend wird hier Funktionalität, die bisher durch Hardwarebausteine verwirklicht wurde, durch immer komplexere Softwarekomponenten ersetzt. Mit der Übernahme von Hardwarefunktionalität durch Software geht häufig eine wesentliche Erweiterung des Funktionsumfangs der betreffenden Systeme einher, wie beispielsweise in der Automobiltechnik, in der sich isolierte, anfangs verhältnismäßig einfache Systeme wie Motorsteuerungen und Antiblockiersysteme zu umfassenden Fahrdynamiksystemen entwickelt haben. Dabei ergibt sich durch die notwendige Vernetzung mehrerer kommunizierender Teilsysteme eine zusätzliche Dimension in der Komplexität dieser Systeme.

Mit diesem, durch Software erst ermöglichten, wachsenden Funktionsspektrum eingebetteter Systeme nimmt die Komplexität des Softwareanteils in einem Umfang zu, daß beim Entwurf der systematische Einsatz von Techniken des Software Engineering zur Komplexitätsbewältigung unumgänglich wird.

1.1 Motivation für die Entwicklung von AutoFOCUS

Angesichts der Verfügbarkeit einer Reihe kommerzieller Entwicklungswerkzeuge für eingebettete Systeme stellt sich die Frage, inwieweit der Entwicklungsaufwand für einen weiteren, akademischen Werkzeugprototyp für diesen Zweck gerechtfertigt ist. Hierfür gibt es unserer Meinung nach eine Reihe von Gründen:

Zuallererst ist festzustellen, daß zum einen bei vielen der kommerziell verfügbaren Werkzeuge besonderes Augenmerk auf pragmatische Aspekte wie komfortable Codegenerierung und einfache Bedienbarkeit (Benutzungsoberfläche, "Assistenten", "Wizards") gelegt wird, während konzeptuelle Grundlagen und semantische Aspekte nur unzureichend behandelt werden. Daneben existiert eine Reihe von Werkzeugen, bei denen besonderer Wert auf die Umsetzung mathematischer Techniken gelegt wird, die jedoch für Systementwickler in der Praxis *zu sehr* an mathematischen Konzepten orientiert sind (dazu siehe auch Abschnitt 1.2, der diese Thematik ausführlicher behandelt). Damit stellt die Verbindung praxisorientierter, auf Anwender ausgerichteter Werkzeugkonzepte und präziser, semantisch fundierter Modellierungskonzepte nach wie vor ein wichtiges Forschungsgebiet dar.

Der Werkzeugprototyp AutoFOCUS unterstützt Kernkonzepte der FOCUS-Methodik [6] zur Entwicklung verteilter Systeme. Durch die Realisierung von AutoFOCUS wurde so eine im Quellcode verfügbare Experimentierplattform zur Evaluierung verschiedener methodischer Konzepte geschaffen.

1.2 Ähnliche Arbeiten und Entwicklungen

Das Spektrum derzeit erhältlicher Werkzeuge im Bereich eingebetteter Systeme erstreckt sich von solchen, die hauptsächlich auf Verifikationstechniken ausgerichtet sind, bis hin zu Werkzeugen, die in erster Linie für Codegenerierung und Simulation entwickelt wurden, wie beispielsweise Atelier B [1], StateMate [10] oder SDT [23]. Im allgemeinen versuchen diese Werkzeuge allenfalls ansatzweise, diese verschiedenen Aspekte zu integrieren, d.h. eine integrierte Kombination aus klar definierten Modellierungskonzepten, einem unterliegenden semantischen Modell als Basis für Verifikationsunterstützung, praxisgerechten, graphischen Notationen zur Systembeschreibung, Prototyping-basierter Validation von Systemen und Codegenerierung zur Systemimplementierung zu vereinigen.

Beispielsweise bietet UML [5] mehrere, im wesentlichen graphische Notationen zur Systembeschreibung. Werkzeuge wie Rational Rose [20] und Rhapsody [14] unterstützen diese Notationen in einem gewissen Umfang. Jedoch fehlt UML bisher ein präzises mathematisches Modell als semantische Basis. Daher sind mit UML bzw. mit den sie unterstützenden Werkzeugen weder Konsistenzprüfungen, die über einfache syntaktische Konsistenz hinausgehen, möglich noch die Verifikation von Systemeigenschaften. Ähnliche Beobachtungen gelten auch für solche Werkzeuge, die formal fundierte Beschreibungstechniken verwenden, wie beispielsweise SDT [23].

Andererseits fehlt bei Werkzeugen, die ihren Ursprung in formalen Techniken haben, wie beispielsweise STeP [4] oder Atelier B [1] häufig methodische Funktionalität wie eine Aufteilung von Systembeschreibungen in wohldefinierte Sichten, oder die Möglichkeit, Systembeschreibungen auf verschiedenen Abstraktionsgraden zu erstellen und zueinander in Beziehung zu setzen.

Das Ziel bei der Entwicklung von AutoFOCUS war es nicht, ein konkurrierendes Werkzeug zu diesen kommerziellen Produkten zu erstellen. Vielmehr sollte untersucht werden, wie die aus vorangegangenen Grundlagenprojekten gewonnenen Erkenntnisse konsequent für die Entwicklung von Software für eingebettete Systeme eingesetzt werden können. Wesentliche Aspekte, die hierbei eine Rolle spielen, sind semantisch sauber fundierte Beschreibungstechniken, die Modularität dieser Beschreibungstechniken zur Strukturierung von Spezifikationen mittels ergänzender Sichten auf verschiedenen Abstraktionsstufen, sowie der methodische Einsatz dieser Techniken für eine integrierte Modellierung und Entwicklung von Systemen. In dieser Hinsicht unterscheidet sich AutoFOCUS auch von anderen Werkzeugen, die für die Entwicklung von Software für eingebettete Systeme eingesetzt werden. So werden beispielsweise in Ansätzen wie bei StateMate unterschiedliche Sichten vermischt (Verhalten und Struktur durch die Verwendung von AND- und OR-Zuständen). Auch fehlt hier die konsequente Nutzung der Möglichkeiten der formalen Basis des Werkzeugs in Form einer Anbindung an Beweiswerkzeuge wie Model Checker oder Theorembeweiser. Während andere Werkzeuge die oben genannte Vermischung von Aspekten vermeiden, wird hier die Verwendung vollständiger, klar spezifizierter Beschreibungstechniken nicht in letzter Konsequenz vollzogen. So ist es beispielsweise bei ObjecTime nötig, die Diagramme zur vollständigen Verhaltensbeschreibung mit Programmcode anzureichern. Auch wurde ROOM [22] bzw. ObjecTime ohne saubere formale Festlegung eines Modells beschrieben; die semantischen Eigenschaften sind letztendlich nur durch die Generierung von ablauffähigen Modellen, also auf sehr implementierungsnaher Ebene beschrieben. Neben möglichen Unklarheiten über Konzepte der Modellierung liegen damit insbesondere nicht die nötigen Voraussetzungen für die Anbindung von formalen Werkzeugen wie Model Checkern oder Theorembeweisern vor.

Die Kombination dieser genannten Aspekte und die daraus resultierenden methodischen Konsequenzen und Möglichkeiten stellen die treibende Kraft bei der Entwicklung von AutoFOCUS dar. AutoFOCUS verwendet eine überschaubare Menge strukturierter Beschreibungstechniken, basierend auf einem gemeinsamen mathematischen Modell. Die Beschreibungstechniken als solche sind jedoch werkzeug- und prozeßunabhängig und können somit mit einer breiten Palette von Entwicklungsprozessen kombiniert werden.

2 Methodischer Hintergrund und Konzepte

Werkzeuge können nur so gut sein wie die ihnen zugrundeliegenden Konzepte. Daher beschreiben wir zunächst die AutoFOCUS zugrundeliegenden Konzepte und die methodische Basis, auf der AutoFOCUS aufbaut. Wir definieren daher

- aus welchen Bausteinen Spezifikationen erstellt werden,
- wie diese Bausteine dargestellt werden,
- wie eine Spezifikation aus diesen Bausteinen erstellt wird
- und wie Teile von Spezifikationen für andere Spezifikationen wiederverwendet werden können.

Diese Punkte werden in den folgenden Abschnitten näher erläutert.

2.1 Modellierungskonzepte für verteilte Systeme

Unabhängig davon, wie Spezifikationen eingebetteter Systeme notationell dargestellt werden, läßt sich eine Reihe von grundlegenden Konzepten identifizieren, die solche Systeme beschreiben. Diese werden im folgenden kurz informell erläutert.

Komponenten sind die Grundbausteine eines eingebetteten Systems. Sie stellen Einheiten dar, die Daten, Struktur und Verhalten umfassen und mit ihrer Umwelt kommunizieren. Komponenten reagieren auf Signale aus ihrer Umwelt mittels Ausgaben. Intern können sie hierarchisch strukturiert sein und eine Menge an untereinander kommunizierenden Subkomponenten umfassen.

Datentypen definieren die Datenstrukturen, die in einem eingebetteten System gespeichert bzw. verarbeitet werden.

Datenelemente sind in Komponenten gekapselt und erlauben die persistente Speicherung von Zustandsinformation. Datenelemente werden durch einfache getypte Zustandsvariablen realisiert.

Ports bilden die Schnittstelle einer Komponente zu ihrer Umwelt. Komponenten lesen Daten von Eingabeports und produzieren Ausgaben auf Ausgabeports. Ports besitzen einen Namen und einen Datentyp.

Kanäle sind gerichtete, mit Namen und Datentyp versehene Kommunikationsverbindungen zwischen den Ports von Komponenten. Durch sie wird die Kommunikationsstruktur, d.h. die Topologie eines eingebetteten Systems festgelegt.

Verhalten beschreibt, wie eine Komponente auf Eingaben aus ihrer Umwelt reagiert. Diese Reaktion ist ein Ergebnis der Eingaben und des internen Zustands der Komponente. Das Verhalten wird zustandsorientiert durch Kontrollzustände, Transitionen und Datenelemente beschrieben.

Im Hinblick auf die unterliegenden mathematischen Formalismen von FOCUS stellen diese genannten Grundkonzepte Abstraktionen sowohl des formalen Modells als auch der konkreten Notationen, mit deren Hilfe sie dargestellt werden, dar. Daher ist dieses konzeptuelle Modell der gemeinsame Kern sowohl der Beschreibungstechniken als auch des formalen Modells.

2.2 Sichten und Notationen: Beschreibungstechniken

Um eine umfassende und strukturierte Spezifikation eines Systems erstellen zu können, sollten Entwickler die Möglichkeit haben, das System aus unterschiedlichen Blickwinkeln (Sichten) und auf verschiedenen Abstraktionsgraden beschreiben zu können. Ähnlich wie andere Werkzeuge bietet AutoFOCUS daher verschiedene Sichten auf ein verteiltes bzw. eingebettetes System und zugehörige Notationen zu deren Beschreibung. Diese sind allerdings im Vergleich zu anderen Beschreibungstechniken konsequent hierarchisch strukturiert und unterstützen methodische Konzepte wie Dekomposition und Verfeinerung bzw. Verbergen von Detailinformation und Abstraktion explizit. Die Notationen sind im einzelnen

- Systemstrukturdiagramme (System Structure Diagrams, SSDs),
- Datentypdefinitionen (Data Type Definitions, DTDs),
- Zustandsübergangsdiagramme (State Transition Diagrams, STDs) und
- (erweiterte) Ereignisdiagramme (Extended Event Traces, EETs),

von denen jede unterschiedliche – teilweise überlappende – Aspekte eines Systems abdeckt. Die Integration dieser Sichten, insbesondere auf der Grundlage eines gemeinsamen mathematischen Modells, führt zu einer integrierten und formalen Spezifikation des Systems.

Wie erwähnt, unterstützt AutoFOCUS Hierarchiekonzepte bei der Entwicklung von Systemen. Abhängig von der gewünschten Granularität können beispielsweise Komponenten, Zustände oder Sichten atomar sein oder ihrerseits aus Subkomponenten, Unterzuständen oder dekomponierten Sichten bestehen. AutoFOCUS erlaubt damit insbesondere auch die Darstellung einer Systembeschreibung auf unterschiedlichen, dem jeweiligen Darstellungszweck angemessenen Detaillierungsgraden.

2.2.1 Systemstrukturdiagramme (SSDs) Ein (verteiltes) System besteht aus seinen Komponenten sowie den Kommunikationskanälen, die zwischen ihnen verlaufen. Ein eingebettetes System kommuniziert mit seiner Umwelt. Um diese statischen Aspekte eines verteilten Systems zu beschreiben, werden in AutoFOCUS Systemstrukturdiagramme (SSDs) verwendet. Graphisch werden SSDs, wie in Abbildung 1[1] gezeigt, ähnlich wie Datenflußdiagramme, durch Graphen mit beschrifteten Rechtecken für Komponenten und beschrifteten Kanten für Kanäle dargestellt. Leere und gefüllte Kreise an den Enden der Kanäle symbolisieren die Ports. Jede Komponente hat einen Namen und eine Menge von Eingabe- und

[1] Die Diagramme und Bildschirmhardcopies in diesem Beitrag stammen aus einer mit AutoFOCUS durchgeführten Fallstudie, einer Ampelsteuerung für eine Fußgängerampel.

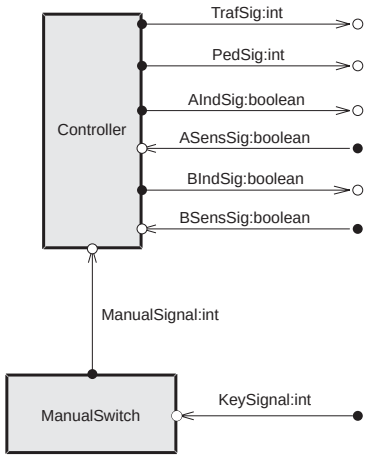

Abb. 1 Systemstrukturdiagramm

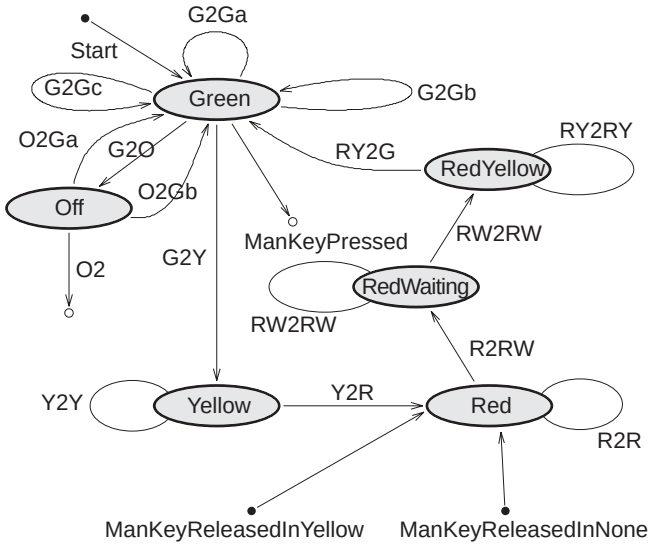

Abb. 2 Zustandsübergangsdiagramm

Ausgabekanälen, die über Ein- und Ausgabeports mit ihr verbunden sind. Jeder Kanal wird durch einen Kanalbezeichner und einen Datentyp, der die Art der über ihn verschickten Nachrichten beschreibt, definiert. Wird kein Wert auf einem Kanal verschickt, so enthält der Kanal einen Default-Wert *nil*.

SSDs beschreiben somit neben der topologischen Sicht eines verteilten Systems auch die syntaktische Schnittstelle jeder Komponente, gegeben durch deren Ports.

Da jede Komponente selbst als verteiltes System gesehen werden kann, ist es möglich, hierarchische Systemspezifikationen zu erstellen. Hier stellen Ports ein Mittel dar, um Beschreibungen zu modularisieren: Ein Port aus der Außensicht einer Komponente ist auch in der Innensicht der Komponente vorhanden, d.h. in dem SSD, das die Innensicht wiedergibt. Somit dienen Ports als Schnittstelle zwischen der Umgebung einer Komponente und ihrer internen Struktur.

In AutoFOCUS kann eine Komponente

- mit einer hierarchischen Unterstruktur, gegeben durch ein SSD,
- mit einer Verhaltensbeschreibung durch Zustandsdiagramme (STDs) oder Event Traces (EETs) und
- mit Komponentendatendeklarationen (CDDs) versehen werden.

Eine Komponentendatendeklaration deklariert lokale Zustandsvariablen einer Komponente durch einen Namen und einen Datentyp für jede Variable. Diese Variablen werden für die Definition des Verhaltens einer Komponente verwendet und geben den Datenzustandsraum der Komponente an.

2.2.2 Daten- und Datentypdefinitionen (DTDs) Die Typen der Daten, die von einem verteilten System verarbeitet werden, werden in AutoFOCUS mit Hilfe einer textuellen Notation, genannt DTD, definiert. Gegenwärtig unterstützt AutoFOCUS zwei Ansätze, um Daten zu beschreiben.

Java: Eine Teilmenge der Basisdatentypen der Programmiersprache Java kann zur Deklaration von Datenelementen verwendet werden. Benutzerdefinierte Datentypen sind hier nicht möglich.

Funktional: Die Basisdatentypen und Typkonstruktoren der Sprache FriscoF [18], einer funktionalen Programmiersprache, die Konzepte von Gofer [16] und Elemente algebraischer Spezifikationssprachen vereint, können verwendet werden, um Datenelemente zu deklarieren bzw. neue Datentypen zu definieren.

Beide Ansätze sind in den Prototypgenerator integriert. Somit ist es möglich, aus Projekten für beide Datentypansätze ablauffähige Prototypen zu generieren.

Wie schon erwähnt, werden Datentypen dazu verwendet, um lokale Komponentenvariablen zu deklarieren, um Datentypen von Kanälen und Ports festzulegen, sowie um lokale Variablen in Transitionen zu deklarieren.

2.2.3 Zustandsübergangsdiagramme (STDs) Ein Zustandsübergangsdiagramm ("State Transition Diagram", STD) beschreibt das Verhalten eines Systems oder einer Komponente in Form der Reaktionen auf Eingaben aus der Umgebung und der daraus resultierenden Ausgaben in die Umgebung. Die Reaktionen hängen vom aktuellen Zustand der Komponente ab. Da ein STD eine erweiterte endliche Zustandsmaschine mit lokalen Datenzuständen der zugehörigen Komponente beschreibt, wird der Gesamtzustand einer Komponente durch den Kontroll- und Datenzustand charakterisiert. Ein Konzept ähnlich zu den Zustandsmaschinen von AutoFOCUS wird in [9] beschrieben. Visuell werden STDs durch Graphen mit ovalen Knoten für die Zustände und annotierte Pfeile für die Transitionen repräsentiert. Abbildung 2 zeigt ein einfaches Beispiel hierfür. Jeder Komponente eines Strukturdiagramms kann ein STD zugewiesen werden. Ähnlich wie bei SSDs kann jeder Zustand eines STDs wiederum in ein weiteres STD dekomponiert werden. Transitionen sind mit den folgenden Bedingungen annotiert:

- Vor- und Nachbedingungen, die durch Prädikate über dem lokalen Datenzustand der zugehörigen Komponente gegeben sind,

– eine Menge von Ein- und Ausgabemustern, welche die Nachrichten, die von den Eingabeports gelesen bzw. auf den Ausgabeports ausgegeben werden, beschreiben und
– eine optionale Beschriftung, die ohne semantische Relevanz als informeller Name der Transition verwendet werden kann.

Zur hierarchischen Dekomposition von Zuständen in STDs wird ein ähnliches Konzept wie bei SSD-Ports verwendet, nämlich Konnektoren, die als Verbindung zwischen Außen- und Innensicht eines Zustands dienen.

Wie erwähnt werden Ein- und Ausgabemuster verwendet, um zu beschreiben, welche Nachrichten an Eingabeports anliegen müssen, damit eine Transition schalten kann, und welche Nachrichten dabei auf die Ausgabeports geschrieben werden.

Ein Eingabemuster besteht aus dem Namen eines Eingabeports und einem Muster, das lokale Komponentenvariablen oder freie Transitionsvariablen enthalten kann. Eine an einem Eingabeport anliegende Nachricht entspricht einem Eingabemuster, falls die Werte der Komponentenvariablen mit den entsprechenden Werten am Port übereinstimmen. Freie Variablen in einem Eingabemuster werden dabei mit den am Port anliegenden Werten belegt.

Ausgabemuster sind Paare von Namen von Ausgabeports und Ausdrücken, welche mit dem Typ des jeweiligen Ports konform sein müssen und über den lokalen und freien Variablen aufgebaut werden können.

In Vorbedingungen dürfen nur lokale Komponentenvariablen verwendet werden, um Prädiakte über dem aktuellen Datenzustand zu formulieren. In Nachbedingungen dürfen Komponentenvariablen und freie Variablen verwendet werden, um Prädikate zu formulieren. Wie in Ein- und Ausgabemustern werden definierte und freie Variablen an ihre aktuellen Belegungen gebunden. Zusätzlich kann für jede definierte Variable x eine gestrichene Variable x' eingeführt werden, um den Wert der Variable nach der Transition zu spezifizieren. Wichtig ist in diesem Zusammenhang, daß im AutoFOCUS-Ansatz alle Eingabeports simultan gelesen werden und auf alle Ausgabeports simultan geschrieben wird.

Eingabemuster in STDs sind vollständig in dem Sinne, daß unspezifizierte Kombinationen von Eingabemustern so interpretiert werden, daß sowohl Kontroll- als auch Datenzustand unverändert bleiben und eine leere Ausgabe geschrieben wird. Ist für einen Eingabeport kein Eingabemuster definiert, so ist der Inhalt des Ports für die Transition irrelevant. Fehlt für den Ausgabeport ein Ausgabemuster, so wird der Ausgabeport mit dem *nil*-Wert belegt.

Als Konsequenz aus dieser ungepufferten Semantik ist das Verhalten von AutoFOCUS-Komponenten näher an hardwareorientierten Beschreibungstechniken, wie beispielsweise StateCharts, als an abstrakten Systembeschreibungssprachen wie SDL (mit Eingabepuffern für jede Komponente).

2.2.4 Ereignisdiagramme (EETs) Neben STDs können auch Ereignisdiagramme (“Extended Event Traces”, EETs) zur Beschreibung des Verhaltens verteilter Systeme verwendet werden. EETs (siehe Abbildung 3) beschreiben Verhalten aus einem komponenten- und kommunikationsorientierten Blickwinkel durch exemplarische Kommunikationshistorien. AutoFOCUS verwendet eine Teilmenge von Elementen des ITU-Standards für Message Sequence Charts (MSCs) [15], ähnlich zu den Konzepten der UML Sequenzdiagramme [5]. Wie auch die anderen graphischen Beschreibungstechniken von AutoFOCUS erlauben EETs den Einsatz hierarchischer Techniken. Sogenannte Boxen können in EETs eingesetzt werden, um darin eine Menge von Sub-EETs zu gruppieren. Da als Ablauf für den Abschnitt wahlweise eines der Sub-EETs eingesetzt werden kann, ermöglichen Boxen sowohl die Strukturierung als auch die Einführung von Verhaltensvarianten in EETs. Überdies können Teilabschnitte von EETs mit sogenannten Indikatoren versehen werden, die optionale bzw. wiederholbare Abschnitte kennzeichnen. Eine vollständige Beschreibung von EETs wird in [21] gegeben. EETs können in verschiedenen Entwicklungsphasen für unterschiedliche Zwecke verwendet werden:

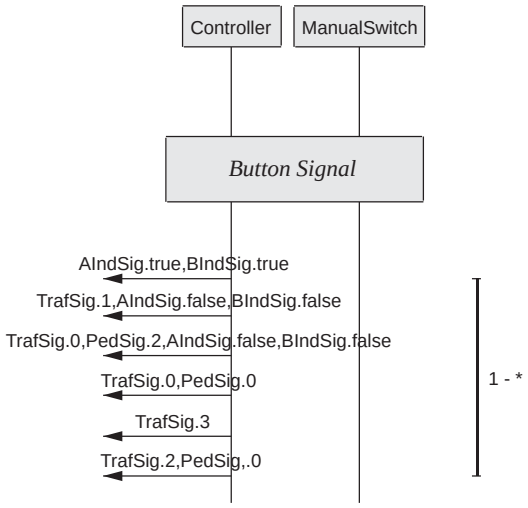

Abb. 3 Ereignisdiagramm

– in frühen Phasen zur Beschreibung elementarer Funktionalität des Systems oder von Verhalten in Fehlersituationen,
– im Verlauf des Entwurfsprozesses können Systemspezifikationen, die durch SSDs, STDs und DTDs gegeben sind, auf die Erfüllung der in EETs geforderten Funktionalität geprüft werden und
– im Rahmen der Validierung eines Systementwurfs können EETs verwendet werden, um die Ausführung eines generierten Prototyps zu visualisieren oder um Fehlerpfade aus einer Model Checker-basierten Verifikation darzustellen.

2.3 Übersicht über den Entwurfsprozeß mit AutoFOCUS

In seiner bisherigen Ausprägung stellt der Systementwurf mit AutoFOCUS einen Top Down-Prozeß dar. Wie in Abbildung

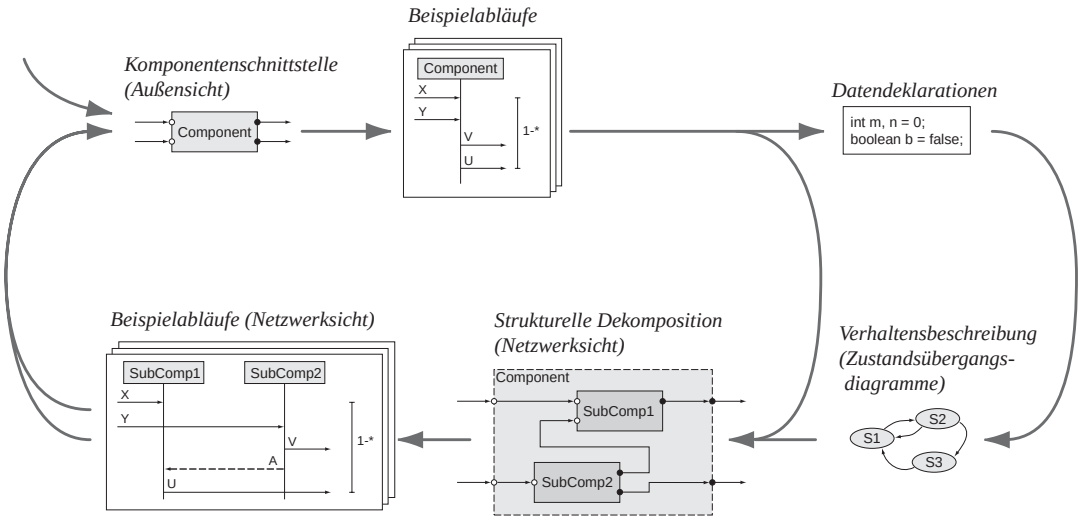

Abb. 4 Systementwurf mit AutoFOCUS im Überblick

4 gezeigt, wird ein System zunächst in seiner Black Box-Sicht als eine mit ihrer Umwelt interagierende Komponente spezifiziert. Die Interaktion erfolgt durch die Schnittstelle mit der Umwelt, definiert durch die Ports der Komponente. Globale Verhaltensaspekte des Systems können hier mit Hilfe von Beispielabläufen in EETs spezifiziert werden. Verhalten kann auch dadurch definiert werden, daß eine explizite Kontrollflußspezifikation in Form eines Zustandsübergangsdiagramms angegeben wird. Falls erforderlich, können im System zu speichernde Daten deklariert werden.

Üblicherweise wird man ein System zu Beginn der Entwicklung zunächst in eine Reihe von Unterkomponenten aufteilen, von denen jeder eine spezifischere Funktionalität zugewiesen wird. Zur Kommunikation untereinander werden diese Subkomponenten über ein Netzwerk von Kanälen miteinander verbunden. Dieser Prozeß der schrittweisen Dekomposition des Systems kann so oft wiederholt werden, wie erforderlich; als Ergebnis ergibt sich eine hierarchische Systemstruktur, die mit jeder Iteration eine feinere Granularität erhält.

Mit der Verteilung der Systemfunktionalität auf verschiedene Komponenten müssen auch die Beispielabläufe, die für die Komponenten in den höheren Abstraktionsebenen spezifiziert wurden, auf die Subkomponenten verteilt und entsprechend projiziert werden (unterer Teil von Abbildung 4). Wiederum kann für jede Komponente in der Dekompositionshierarchie eine Kollektion lokaler Datenelemente deklariert werden, ebenso wie für jede Komponente eine Kontrollflußspezifikation durch ein STD angegeben werden kann (rechter Teil von Abbildung 4).

Für eine komplette Fallstudie, die diesen Entwurfsprozeß anhand eines Beispiels zeigt, verweisen wir auf [11].

2.4 Wiederverwendung von Spezifikationen und Komponenten

Innerhalb des skizzierten Entwicklungsprozesses ist es selbstverständlich wünschenswert, daß nicht alle Komponenten eines Systems, die im Lauf der Dekomposition definiert werden, neu spezifiziert werden müssen, sondern früher spezifizierte Komponenten wiederverwendet werden können. Aus einer bereits vorhandenen Bibliothek von Komponenten sollten geeignete in neue Systemspezifikationen integriert werden können, eine Vorgehensweise, die in der Natur des komponentenbasierten Modellierungsansatzes von AutoFOCUS liegt. Jedoch ist Wiederverwendung von Komponentenspezifikationen in der momentanen Implementierung von AutoFOCUS aus primär technischen Gründen nicht in so geradliniger Weise wie wünschenswert möglich: AutoFOCUS verwendet momentan noch ein dokumentenbasiertes Repository (siehe auch Abschnitt 3). Auf dieser Basis ist ein konzeptbasierter Ansatz[2] für Wiederverwendung von Komponenten schwierig zu realisieren, wohingegen die Wiederverwendung auf Dokumentenbasis naturgegeben sehr einfach zu verwirklichen ist.

Diesem Defizit wird aktuell durch die Umstellung auf ein modellbasiertes Repository Rechnung getragen, wie in Abschnitt 8 beschrieben.

[2] Unter konzeptbasiert verstehen wir in diesem Zusammenhang einen Modellierungsansatz, bei dem ein abstraktes, syntaktisches Modell eines Systems erstellt wird. Komponenten enthalten dabei wesentlich mehr an Information als in separaten SSD-Dokumenten oder einem STD-Dokument erfaßt wird. Hier muß, um eine Komponente wiederzuverwenden, Information aus verschiedenen Dokumenten “herausgeschnitten” werden.

3 AutoFOCUS Spezifikationswerkzeuge

3.1 Client/Server-Architektur

Aufgrund ihrer Komplexität werden insbesondere verteilte Systeme auch häufig in Teams mit mehreren Entwicklern entwickelt. AutoFOCUS besitzt daher eine Client/Server-Architektur mit einem zentralen Repository, in dem alle Entwurfsdokumente verwaltet werden. Mehrere Clients können auf dieses Repository über ein Netzwerk zugreifen. Somit können mehrere Entwickler simultan auf Spezifikationsdokumente zugreifen und diese bearbeiten. Durch die Implementierung der AutoFOCUS-Clients in Java können diese auf allen gängigen Systemplattformen eingesetzt werden.

Versions- und Zugriffskoordination Spezifikationen werden wiederholt überarbeitet, insbesondere in den frühen Phasen der Systementwicklung. Um dies zu unterstützen, bietet AutoFOCUS eine Versionsverwaltung sowohl auf Dokumenten als auch auf ganzen Projekten. Die Koordination konkurrierender Zugriffe auf Dokumente im Repository erfolgt nach dem üblichen Verfahren beliebig vieler gleichzeitiger Lesezugriffe bei nur einem zu einem Zeitpunkt möglichen Schreibzugriff.

3.2 AutoFOCUS Client

Auf der Client-Seite ist die gesamte Funktionalität von AutoFOCUS in einer graphischen Benutzungsoberfläche zugänglich. Teile davon sind in Abbildung 5 gezeigt. Ein Projekt-Browser erlaubt Entwicklern den Zugriff auf die Projekte im Repository sowie auf die nach Dokumentenklassen gruppierten, hierarchisch strukturierten Spezifikationsdokumente.

Für jede der in Abschnitt 2.2 eingeführten Beschreibungstechniken bietet AutoFOCUS entsprechende graphische Editoren, welche ebenfalls in Abbildung 5 gezeigt werden. Alle Editoren haben eine einheitliche Benutzungsoberfläche, mit Clipboard-Funktionalität, grundlegender Unterstützung für graphische Operationen und eine einfache Form automatischer Wegewahl für Kanäle bzw. Transitionen. Um mit AutoFOCUS erstellte Diagramme auch außerhalb des Werkzeugs, beispielsweise zur Einbindung in Dokumentationen, zu verwenden, können Diagramme in Encapsulated PostScript-Format exportiert werden.

Dokument-basierte Spezifikation In AutoFOCUS besteht ein Projekt, das ein in Entwicklung befindliches System repräsentiert, aus einer Menge von Spezifikationsdokumenten, die wiederum Sichten, dargestellt durch die beschriebenen Notationen, repräsentieren. Daher entspricht jede Sicht einer korrespondierenden Klasse von Spezifikationsdokumenten oder Diagrammen. Zusammen ergeben diese verschiedenen Diagramme eine vollständige Systembeschreibung im aktuellen Stand ihrer Entwicklung. Zugriffs- und Versionskontrolle werden im Repository auf Ebene dieser Dokumente durchgeführt. Zur Wiederverwendung von Spezifikationsdokumenten können diese aus mehreren Projekten referenziert werden. Auf Projekten wird ebenfalls eine Versionsverwaltung durchgeführt. So ist in AutoFOCUS eine Projektversion durch die Einzelversionen aller enthaltenen Spezifikationsdokumente gekennzeichnet.

Integrierte Sichten Aus Benutzersicht sind die Dokumente eines Entwicklungsprojekts, obwohl an sich separate Dokumente, eng gekoppelt, sowohl vertikal entlang der Dekompositionshierarchien als auch horizontal entlang der Beziehungen zwischen den Dokumenten verschiedener Klassen. Beispielsweise kann ein Zustandsdiagramm mit einer Komponente in einem Strukturdiagramm verknüpft werden, wodurch ausgedrückt wird, daß dieses Zustandsdiagramm das Verhalten der Komponente beschreibt. Aus Benutzersicht bietet AutoFOCUS entlang dieser Beziehungen schnelle und intuitive Navigationsmöglichkeiten.

4 Konsistenzsicherung von Spezifikationen

Übersteigt eine Spezifikation die Größe eines Spielbeispiels, so ist die Spezifikationsinformation über mehrere Sichten und Abstraktionsebenen verteilt. Dies stellt eine wesentliche Quelle für Fehler und Inkonsistenzen dar. Hier können bereits einfache syntaktische Überprüfungen eine wertvolle Hilfe darstellen. Da AutoFOCUS unterschiedliche *Arten* von Sichten ebenso wie *hierarchische* Sichten verwendet, ist es umso wichtiger, sicherzustellen, daß die über die Sichten verteilte Information wohldefiniert oder in methodischer Hinsicht “konsistent” ist. Daher werden Konsistenzüberprüfungen angeboten, um das Zusammenpassen der bearbeiteten Sichten sicherzustellen.

4.1 Der Konsistenzbegriff in AutoFOCUS

Unter dem Begriff Konsistenz werden hier unter anderem verschiedene Klassen syntaktischer Korrektheitsbedingungen zusammengefaßt wie

Grammatikalische Korrektheit: Die Sichten respektieren die syntaktischen Regeln für die textuelle bzw. die graphgrammatischen Regeln für die graphische Darstellung.

Korrektheit der Sichtschnittstellen: Falls eine Sicht hierarchisch in eine andere Sicht eingebettet wird, müssen die Sichten verträgliche Schnittstellen aufweisen (z.B. verträgliche Schnittstellen für Komponenten und darin enthaltene Systeme).

Definiertheit: Falls in einer Sicht Elemente aus einer anderen Sicht verwendet werden, so müssen diese in der entsprechenden Sicht verwendet werden (z.B. Kanaltypen in SSDs oder STDs).

Typkorrektheit: Der für die Anwendung eines Elements benötigte Typ muß mit dem Elementtyp übereinstimmen (z.B. Port und Wert in STD-Portmuster).

Vollständigkeit: Alle “notwendigen” Sichten eines Projektes liegen vor (z.B. im Falle der Simulation hat jede Komponente ein Sub-SSD oder ein STD).

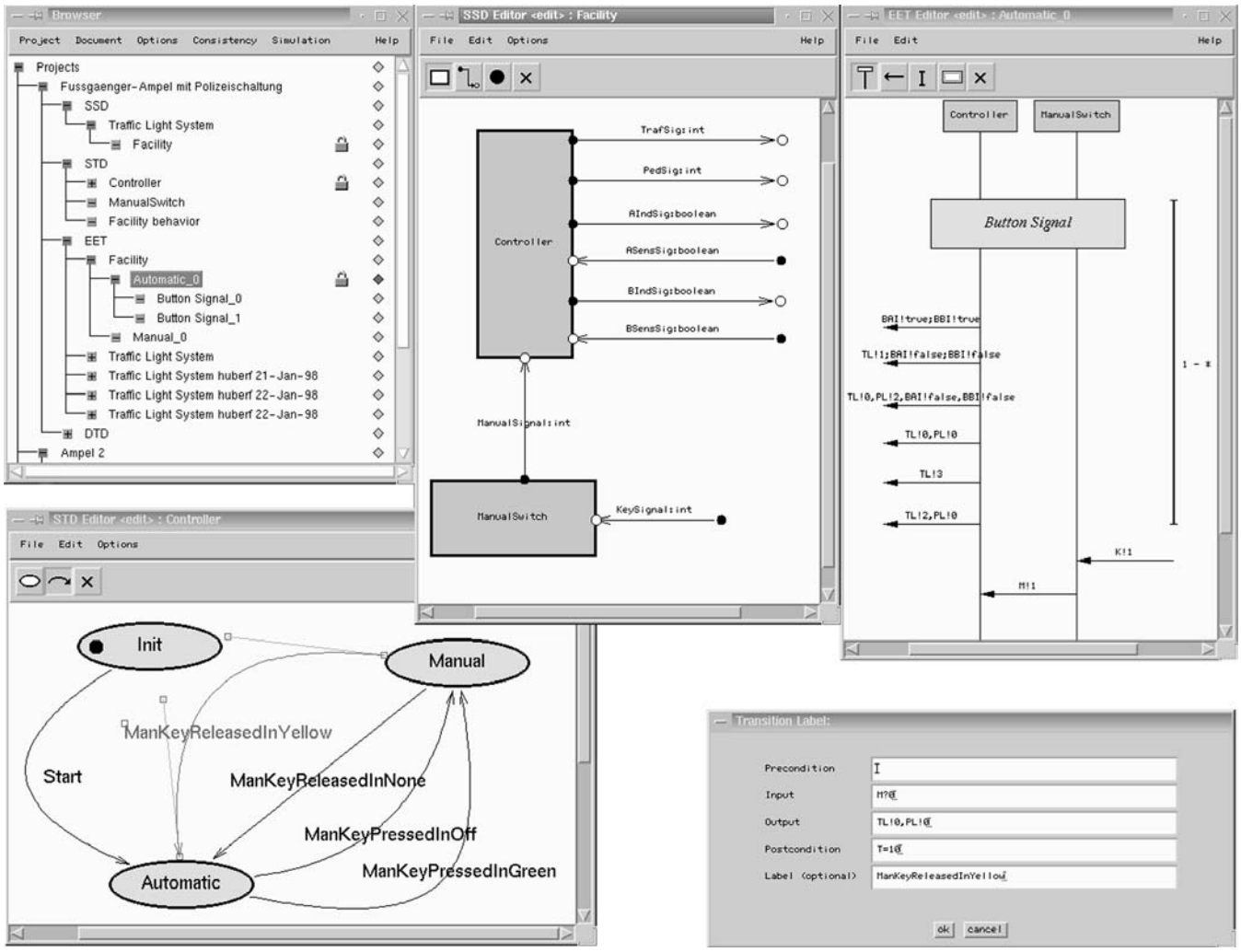

Abb. 5 Die AutoFOCUS Client-Anwendung

Eine einfache Form der Konsistenz ist beispielsweise die grammatikalische oder die Typkorrekheit. Einfache Konsistenzbedingungen für ein SSD sind:

- Jede Komponente hat einen nichtleeren Namen.
- Jeder Port ist an einen Kanal gebunden.
- Jeder Kanal ist pro Richtung an einen Port gleichen Typs gebunden.

Diese Bedingungen lassen sich leicht unter Verwendung von Prädikatenlogik erster Stufe formalisieren, mittels Einführung geeigneter Individuenbereiche für die elementaren Objekte wie Identifikatoren, Komponenten, Ports oder Konnektoren, Richtungen, sowie entsprechender Funktionen wie `name_of`, `type_of`, oder `direction_of`. So läßt sich die letzte Bedingung formalisieren als

$$
\begin{aligned}
&\forall chan : Channel.\\
&\exists icon, ocon : Connector.\exists type : Type.\\
&\quad \texttt{channel\_of}(icon) = chan\\
&\wedge\ \texttt{channel\_of}(ocon) = chan\\
&\wedge\ \texttt{has\_type}(icon, type) \wedge \texttt{has\_type}(ocon, type)\\
&\wedge\ \texttt{has\_type}(chan, type)
\end{aligned}
$$

Weiterhin verwenden wir Überprüfungen ähnlich wie sie wie im Falle des Bindeprozesses von Programmbibliotheken verwendet werden. So lassen sich beispielsweise die Schnittstellenkorrektheit, die Definiertheit oder die Vollständigkeit prüfen. Beispiele für diese Bedingungen, die mehr als eine Sicht benötigen, sind

- Zu jedem Port der Innensicht einer Komponente, der eine Schnittstelle zur Umgebung beschreibt, existiert ein einzelner Port der Komponente in der Außensicht mit dem gleichen Namen, dem gleichen Typ und entgegengesetzter Richtung.
- Hat eine Komponente eine Innensicht, so existiert zu jedem Port der Komponente ein entsprechender einzelner Port in der Innensicht mit dem gleichen Namen, Type und entgegengesetzter Richtung.

Wieder lassen sich diese Bedingungen wie oben formalisieren. Hierzu müssen jedoch Sichten als Individuen und entsprechende Funktionen in die Sprache aufgenommen werden.

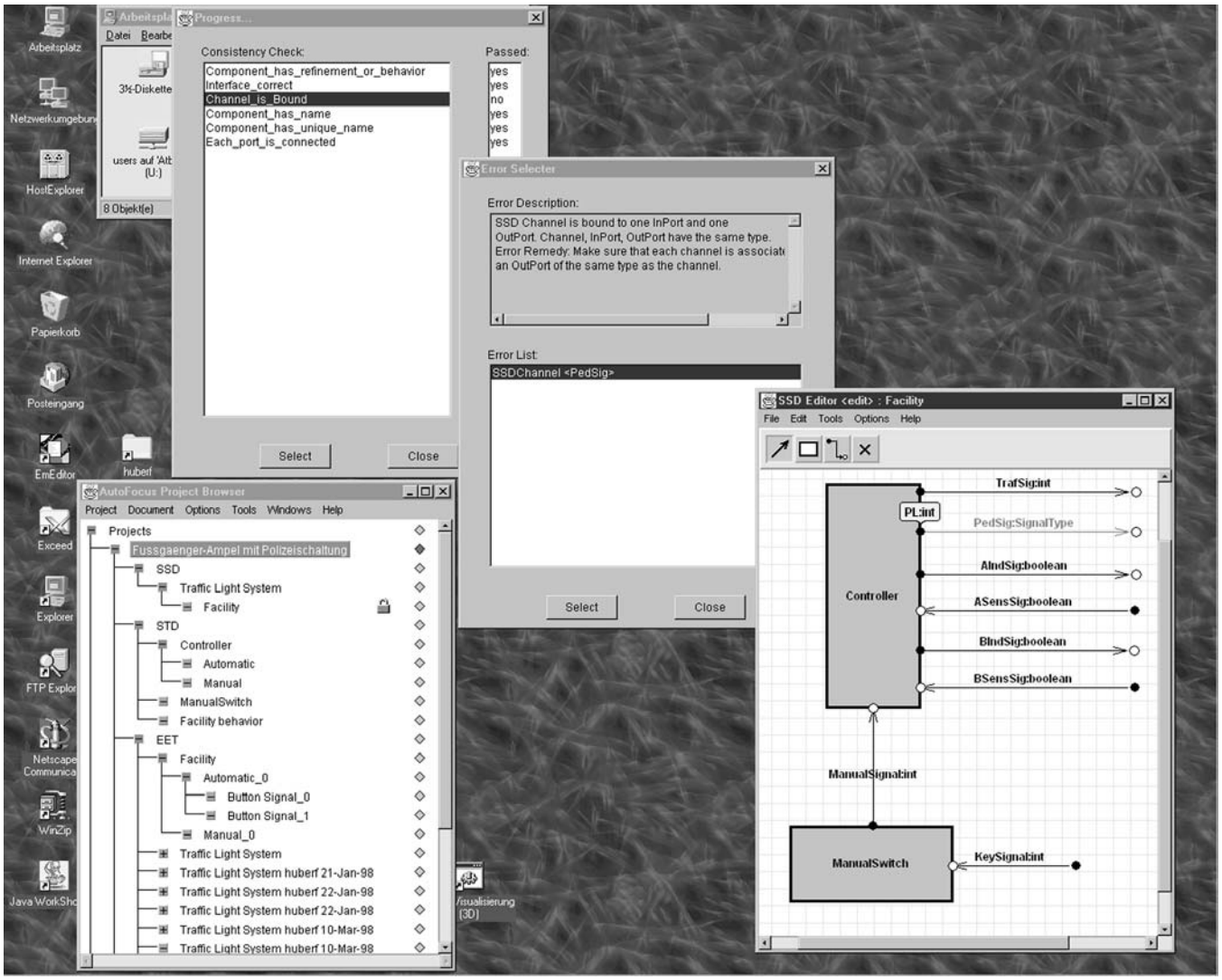

Abb. 6 Der Konsistenzsicherungsmechanismus

4.2 Definition von Konsistenzbedingungen

In AutoFocus werden Konsistenzbedingungen in einer deklarativen textuellen Notation ähnlich zur Prädikatenlogik erster Stufe mit einfacher Typisierung erstellt. Dieser Ansatz erlaubt, im Gegensatz zu einer fest vorprogrammierten Verwendung von Konsistenzbedingungen, wie es bei anderen Ansätzen der Fall ist, die einfache Erweiterung der Konsistenzbedingungen. Der eigentlichen Definition werden darüberhinaus zusätzliche Informationen in Form einer informellen Erklärung der Konsistenzbedingung sowie Hinweisen zur Behebung der Inkonsistenz hinzugefügt. Die Sprachelemente der textuellen Darstellung der Konsistenzbedingungen sind

- Quantoren (Allquantor und Existenzquantor),
- Selektoren zum Zugriff auf Elemente der Spezifikation, sowie
- logische Operatoren und der Gleichheitsoperator.

Das folgende Beispiel zeigt die im Werkzeug eingesetzte Syntax zur Definition der für die Simulation benötigte Vollständigkeitsbedingung:

```
forAll c: Component .
        (exists ssd: SSDView .
                decomposition(c, ssd)) OR
        (exists std: STDView .
                behavior(c, std))
```

Diese Konsistenzbedingung stellt sicher, daß jede Komponente eines SSDs entweder in Unterkomponenten in einem eigenen SSD aufgebrochen werden kann, oder ein STD zur Beschreibung ihres Verhaltens zugeordnet hat.

4.3 Integration der Inkonsistenzbehebung

In AutoFocus wird die Konsistenzprüfung im Projekt-Browser gestartet. Da im allgemeinen während der Projektentwicklung nicht alle Konsistenzbedingungen erfüllt sein

sollen, kann der Benutzer aus der Liste aller vorhandenen Bedingungen eine geeignete Teilmenge auswählen. Das Werkzeug führt dann die gewählten Überprüfungen aus und zeigt im Anschluß die Liste aller gefundenen Inkonsistenzen einschließlich aller betroffenen Sichten. Abbildung 6 zeigt die Bedienoberfläche des Konsistenzsicherungsmechanismus. Aus der Liste heraus können dann direkt die zur Verletzung der Konsistenz führenden Sichten angezeigt werden, wobei die entsprechenden Elemente hervorgehoben sind.

5 Validierung durch Prototyping und Simulation

In der industriellen Anwendung ist die Verwendung von Simulationen und Prototypen das wesentliche Mittel zur Validierung im Entwicklungsprozeß. Durch die Visualisierung von Abläufen auf der Ebene der im Designprozeß verwendeten Beschreibungstechniken können Eigenschaften eines Systems beobachtet und überprüft werden, wie das Verhalten eines Systems bei bestimmten Signalen aus der Umwelt. In Verbindung mit formalen Entwicklungstechniken können Simulation und Prototypen eingesetzt werden, um in ersten Iterationen eines zyklischen Entwicklungsansatzes die Ausgangsbasis für die Entwicklung eines zuverlässigen Systems zu liefern; in den weiteren Schritten können dann formale Techniken wie beispielsweise Model Checking eingesetzt werden.

Zur Unterstützung dieses Ansatzes bietet AutoFOCUS die Komponente SIMCENTER mit den Möglichkeiten

- aus dem Gesamtsystem oder Ausschnitten ausführbare Prototypen zu generieren,
- diese Prototypen in einer Simulationsumgebung auszuführen,
- die Simulationsabläufe mit den Beschreibungstechniken des Entwicklungsprozesses zu visualisieren,
- sowie zusätzliche externe Systeme, wie Multimediafrontends oder externe Hardware an die Simulationsumgebung anzukoppeln.

5.1 Codegenerierung

Wie in [12] beschrieben wird zur Generierung der Prototypen eine schematische Umsetzung in Java-Code verwendet. Dieser Ansatz erlaubt das dynamische Laden des generierten und übersetzten Codes zu den ebenfalls in Java realisierten Komponenten von AutoFOCUS und SIMCENTER. Zusätzlich zur so erzielten Plattformunabhängigkeit spricht für die Verwendung von Java weiterhin die Entwicklung von Java als Plattform für eingebettete Systeme, insbesondere für Java-basierte Hardware.

Für die Generierung wird jeweils die Konsistenz der Spezifikation, wie in Abschnitt 4 beschrieben, vorausgesetzt. Die Codegenerierung basiert auf der in Abschnitt 2.2.3 sowie in [12] beschriebenen synchronen Semantik. Im Gegensatz zu dem dort dargestellten, lose gekoppelten Ansatz wird jedoch

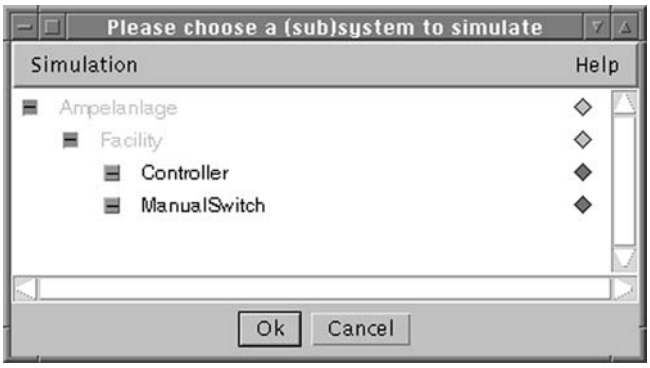

Abb. 7 Der Browser zur Auswahl der zu simulierenden Komponenten

in SIMCENTER zusätzlich eine zentrale Synchronisierungskomponente erzeugt, die die globale Taktung der einzelnen Komponenten des entwickelten Systems sicherstellt.

Nicht in allen Fällen ist die Simulation des gesamten entwickelten Systems wünschenswert. Um auch die Simulation von Systemausschnitten zu unterstützen, erlaubt AutoFOCUS dem Benutzer, die Granularität der Simulation durch Auswahl der zu simulierenden Komponenten zu wählen. AutoFOCUS verwendet hierzu einen hierarchischen Auswahlmechanismus, der die mittels SSDs beschriebene hierarchische Systemstruktur wiedergibt (siehe Abbildung 7). Hierbei ist die Wahl zwischen einer Komponente und deren Subsystem im Auswahlwerkzeug nur dann möglich, wenn beiden – wie in der in Abschnitt 4.2 gezeigten Vollständigkeitsbedingung gefordert – ein Verhalten zugeordnet ist und somit beide simulierbar sind.

5.2 Interaktive Animation und Visualisierung

Für die nutzergerechte Simulation ist die Darstellung der Simulationsinformation auf einer angemessenen Beschreibungsebene von zentraler Bedeutung. Da in AutoFOCUS auch die Implementierung nicht auf Code- sondern auf SSD/STD-Ebene beschrieben wird, werden in SIMCENTER diese Beschreibungstechniken zur Animation verwendet. Entsprechende SIMCENTER-Komponenten stellen Benutzern die gleiche Betrachtungs- und Funktionsweise zur Verfügung, wie sie sie von den Editoren gewohnt sind:

SSD-Animatoren zur Visualisierung der auf den Kanälen transportierten Nachrichten

STD-Animatoren für die Darstellung der aktuellen Zustände und der aktiven Transitionen

CDD-Animatoren für die Anzeige der aktuellen Belegungen von lokalen Variablen von Komponenten

EET-Animatoren für die Aufzeichnung der aktuellen Simulation als Laufzeitprotokoll

Abbildung 8 zeigt diese Animatoren während des Ablaufs des Ampelbeispiels.

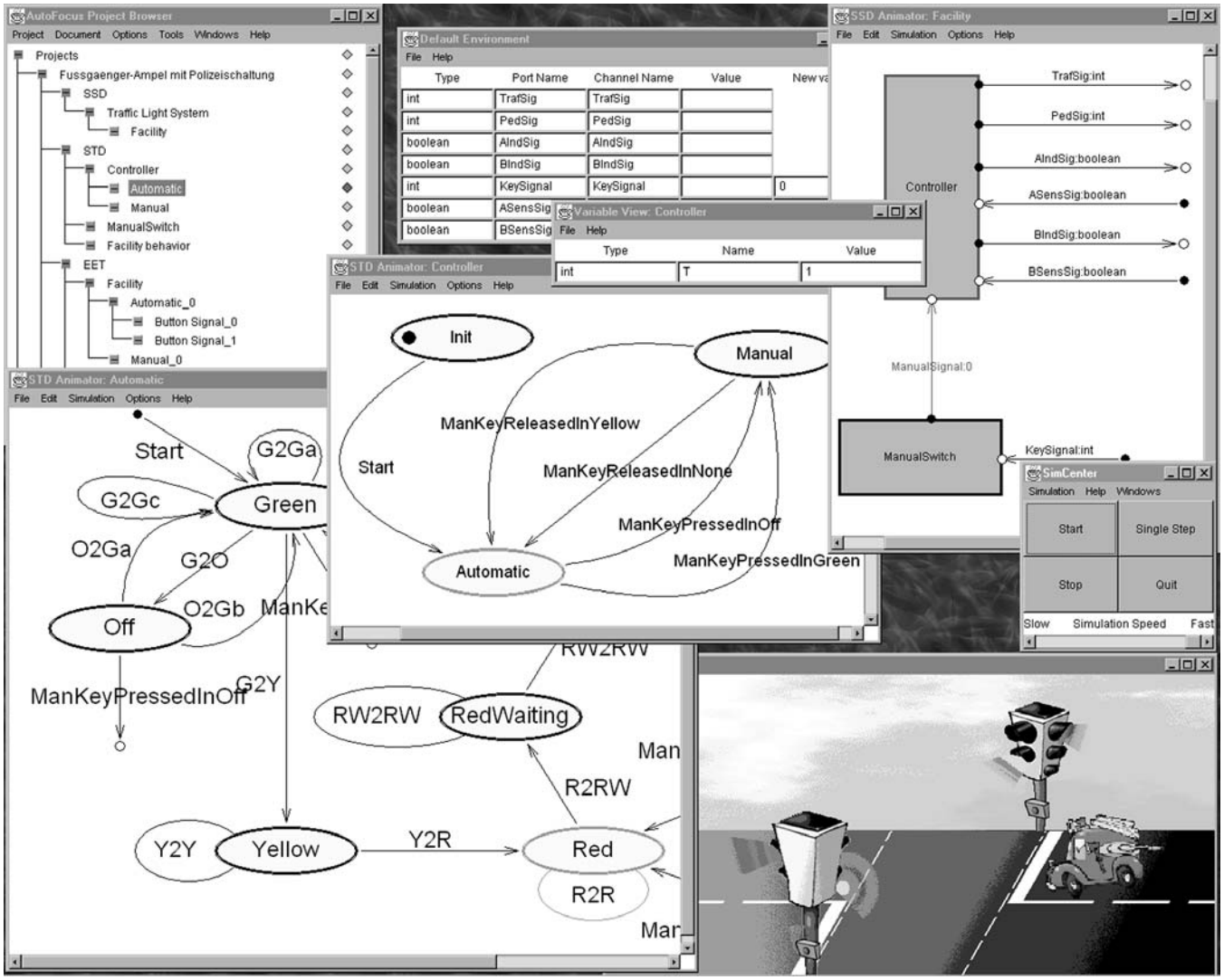

Abb. 8 Eine laufende Simulation mit mehreren Animatoren und einer externen Visualisierung

5.3 Simulationsumgebung

Gerade im Bereich der eingebetteten Systeme ist eine Simulation ausschließlich auf der Ebene der Beschreibungstechniken wie in Abschnitt 5.2 dargestellt nicht immer ausreichend. Hier ist es oft nötig, andere Formen der Darstellung der Interaktionen mit der Systemumgebung anzubieten, um Anwendungsexperten einen unmittelbaren Zugang zur Simulation zur Verfügung zu stellen.

Standardumgebung Die Standardumgebung stellt Benutzern eine Schnittstellensicht auf das System in Form einer Tabelle zur Verfügung. Während in der Tabelle die auf den Ausgabeports zur Verfügung stehenden Nachrichten angezeigt werden, können die den Eingabeports zugeordneten Felder mit Werten belegt werden und damit entsprechende Nachrichten von der Umgebung an das System gesendet werden.

Fremdwerkzeuge Zur weiteren Visualisierung können unter Verwendung einer allgemeinen Kommunikationsschnittstelle Fremdwerkzeuge an SIMCENTER angeschlossen werden. Dies ermöglicht es, die Ausgaben des Systems vom Fremdwerkzeug darstellen zu lassen und umgekehrt entsprechende Benutzerinteraktionen mit dem Fremdwerkzeug in Signale für das simulierte System umzusetzen. Im Fall der Fußgängerampel wurde beispielsweise das gesamte System auf der Ebene der Ampelbenutzersicht visualisiert. Dazu wurde das Multimediawerkzeug *FormulaGraphics* [8] unter Verwendung eines Schnittstellenpakets eingesetzt und lieferte die in Abbildung 8 unten rechts gezeigte Interaktionskomponente.

Durch die Verwendung der Remote Method Invocation (RMI)-Schnittstelle von Java erlaubt es SIMCENTER, unterschiedliche Plattformen für SIMCENTER und die Simulationsumgebung zu verwenden und diese mittels Internet-Protokoll zu verbinden. Diese Entkopplung ermöglicht es, unter Verwendung entsprechender Schnittstellentreiber SIMCENTER an eine Vielzahl von Umgebungsplattformen anzubinden.

5.4 Exportfunktionalität

Zusätzlich zur eingebauten Java-Codegenerierung wurde in AutoFOCUS eine allgemeine Exportschnittstelle realisiert. Durch den Export in Tabellen- bzw. Listendarstellung sowie Perl-Datenstrukturen [24] können zusätzliche Codegeneratoren angeschlossen werden. Unter Verwendung dieser Schnittstelle wurden bereits Generatoren für C-Code aus ausführbaren Spezifikationen realisiert sowie die Umsetzung in die Eingabesprache für die Model Checker SMV [7] und STeP [4]. Letztere wurde beispielsweise für den Nachweis von Eigenschaften in der Ampelfallstudie eingesetzt (siehe Abschnitt 7.1). Basierend auf der gleichen Schnittstelle existiert darüberhinaus eine Umsetzung in *Tiger-BASIC*, die die Generierung von Code zum Laden in das Microcontrollersystem *Basic Tiger* [25] erlaubt. Dieser Code wurde beispielsweise zur Steuerung des in Abschnitt 7.2 erwähnten Aufzugmodells verwendet.

6 Verifikation: Model Checking

Neben dem Nachweis der konzeptuellen Konsistenz einer Spezifikation sowie deren Validierung mittels Simulation ist für eine umfassende formale Entwicklung, wie sie in AutoFOCUS angestrebt wird, auch die Sicherung der semantischen Konsistenz notwendig. Für den praktischen Einsatz sind dabei in erster Linie vollautomatische Techniken wie das Model Checking von Bedeutung. Neben der Klärung der in AutoFOCUS verwendeten Konzepte kommt hier die semantische Fundierung der Beschreibungstechnik zum Einsatz, nämlich in Form der bedeutungserhaltenden Abbildung der spezifizierten Systemsichten auf das dem Model Checker zugrundeliegende Modell. In Abschnitt 5.4 wurde bereits die Anbindung von Model Checkern mittels der Exportschnittstelle erwähnt. Entsprechend des in Abschnitt 1.1 beschriebenen Ansatzes ist für die vollständige Durchgängigkeit von AutoFOCUS neben einer Anbindung zusätzlich eine für den Benutzer transparente Einbettung notwendig. Ebenso wie im Fall der Simulation sind alle für die Verifikation notwendigen Informationen mittels der AutoFOCUS-Beschreibungstechniken ausdrückbar; darüberhinaus sind alle nötigen Benutzeraktionen direkt in AutoFOCUS ausführbar.

Entsprechend der in AutoFOCUS eingesetzten Beschreibungstechniken und der Vorgehensweise beim Systementwicklungsprozeß (vgl. Abschnitte 2.2 und 2.3) können zwei Formen semantischer Inkonsistenzen auftreten:

Szenarieninkonsistenz: Mittels EETs beschriebene Abläufe eines Systems können von der mittels SSDs und STDs beschriebenen Realisierung nicht ausgeführt werden.

Verfeinerungsinkonsistenz: Die mittels eines STDs beschriebenen Abläufe einer Komponente sind keine Obermenge der Abläufe der Realisierung der Komponente mittels eines Sub-SSDs und zugehöriger STDs.

Für beide Formen der semantischen Inkonsistenz stehen entsprechende vollautomatische Überprüfungsmechanismen, wie im folgenden beschrieben, zur Verfügung.

6.1 Szenarienkonsistenz

Der Nachweis der Szenarienkonsistenz spielt besonders in der frühen Phase des formalen Entwicklungsprozesses eine Rolle. Hierbei wird das Verhalten eines Systems, beschrieben mittels SSDs und STDs, mit möglichen beispielhaften Abläufen in Form von EETs verglichen. Werden dabei die EETs als positive Beispiele aufgefaßt, so liegt eine semantische Inkonsistenz vor, wenn diese Beispielabläufe nicht vom System ausgeführt werden können, also nicht in der Menge der Abläufe des Systems enthalten sind. Neben der Verwendung von Szenarien als positive Anforderung an ein System können diese auch als Negativbeschreibung zur Charakterisierung unerlaubter Abläufe eingesetzt werden. Entsprechend liegt hier eine semantische Inkonsistenz vor, wenn einer der mittels EETs beschriebenen Abläufe in der Menge der Systemabläufe enthalten ist.

Da EETs Szenarien sowohl vollständig – beispielsweise durch unbeschränkte Abläufe mittels Wiederholungsoperatoren – als auch partiell – also mit einer nicht im EET dargestellten Fortsetzung – beschreiben, wird für ihren Einsatz als positive und negative Anforderungen jeweils eine Überprüfung für vollständige und partielle Szenarien angeboten.

6.2 Verfeinerungskonsistenz

Ein wesentlicher Designschritt in der Entwicklung eines verteilten Systems liegt in der Realisierung einer abstrakten Komponente durch ein entsprechendes Subsystem. Im formalen Entwicklungsprozeß, wie er von AutoFOCUS unterstützt wird, können auch hier semantische Inkonsistenzen auftreten. Hierzu ist es nötig, daß sowohl für die abstrakte Komponente – durch die Angabe eines STDs – als auch für deren Realisierung mittels eines Subsystems – durch die Angabe von STDs für alle Subkomponenten – deren Verhalten vollständig beschrieben ist. Eine semantische Inkonsistenz liegt vor, wenn die Realisierung der Komponente Verhalten aufweist, das im Falle der abstrakten Komponente nicht auftreten kann. Technisch bedeutet dies, daß die Menge der Abläufe der Realisierung einer Komponente nicht in der Menge der Abläufe des STDs der abstrakten Komponente enthalten sind.

6.3 Realisierung

Zur Überprüfung der Szenarien- und der Verfeinerungskonsistenz wird der relationale μ-Kalkül Model Checker μcke [3] verwendet. Die wesentlichen Grundlagen der Formalisierung von SSD, STDs und EETs im relationalen μ-Kalkül werden in [13] erläutert. Die Benutzerschnittstelle basiert auf ähnlichen Konzepten wie im Falle des Auswahlmechanismus der SIMCENTER-Simulation und wird detailliert in [2] erläutert. Zusätzlich zur Überprüfung der Konsistenz liefert die Modellprüfung mittels Model Checkern im Falle der Inkonsistenz ein entsprechendes Gegenbeispiel. Für eine vollständige Integration ist hierzu noch die Umsetzung von solchen Gegenbeispielen in EETs notwendig, um so dem Benutzer ein transparentes Arbeiten zu ermöglichen.

7 Fallstudien

Die Praxistauglichkeit von AutoFOCUS wurde in einer Reihe von Fallstudien erprobt, von denen im folgenden einige aufgeführt sind.

7.1 Steuerung einer Ampelanlage

Als eines der ersten Beispiele wurde die Steuerung einer Fußgängerampel verwendet. Auszüge der Fallstudie wurden in den vorherigen Abschnitten zur Illustration eingesetzt. Die vollständige Beschreibung findet sich in [11]. Diese Ampelsteuerung wurde auch als Testbeispiel zur Realisierung der Simulationskomponente SIMCENTER verwendet. Dabei bestätigte sich, daß insbesondere eine gut realisierte externe Visualisierung sehr zur Kommunizierbarkeit eines Systementwurfs beiträgt, was auch in den weiteren Fallstudien deutlich wurde.

Selbst für ein relativ einfaches System wie diese Ampelsteuerung ist die Erstellung einer korrekten Spezifikation nicht trivial. Die Möglichkeit, den Systemablauf auf Spezifikationsebene mitzuverfolgen, stellte ein komfortables Mittel dar, um Fehler in der Spezifikation schnell zu lokalisieren und zu beseitigen. Gerade unbeabsichtigt nichtdeterministisches Verhalten des Systems, das durch Unachtsamkeiten bei der Modellierung spezifiziert wurde, konnte mit der Simulation, die bei Transitionen optional auf Nichtdeterminismus hinweist, sehr schnell lokalisiert werden.

Mit Hilfe der in Abschnitt 5.4 angesprochenen Exportschnittstelle wurde aus der Spezifikation der Ampel sowohl Code für den *BASIC Tiger*, und damit ein ohne die SIMCENTER-Umgebung ausführbares eingebettetes System erzeugt als auch Code zur Verifikation verschiedener Eigenschaften mit einer Reihe von Model Checkern. Hierbei zeigte sich wie erwartet, daß nicht alle erprobten Model Checker mit der verhältnismäßig hardwarenahen AutoFOCUS-Semantik gleich gut koppelbar sind.

7.2 Aufzugsteuerung

Im Rahmen eines Studentenprojekts wurden AutoFOCUS und SIMCENTER zur Entwicklung einer Liftsteuerung eingesetzt. Einer der Schwerpunkte lag dabei auf der Kopplung von SIMCENTER mit dem Visualisierungswerkzeug FormulaGraphics sowie mit einem in “Fischer Technik” realisierten Modell des Lifts. Insbesondere die Frage, inwieweit die Javabasierte Simulation geeignet ist, ein reales Hardwaresystem mit schwachen Realzeitanforderungen im Sekundenbereich und knapp darunter zu steuern, war bei dieser Fallstudie von Interesse. Die prinzipielle Eignung von SIMCENTER für derartige Aufgaben konnte gezeigt werden, jedoch wurde – kaum überraschend – deutlich, daß dieser Zeitbereich auch bereits die Grenze der Realzeitanforderungen, die mit SIMCENTER noch handhabbar sind, darstellen. Nichtsdestotrotz stellt die Option, für solche Systeme einen hardwarebasierten Prototypen direkt ansteuern zu können, eine interessante Alternative zu einer rein softwarebasierten Visualisierung dar.

Aus der Spezifikation der Liftsteuerung wurde darüberhinaus *Tiger-BASIC*-Code erzeugt, mit dessen Hilfe das Liftsystem auch ohne SIMCENTER-Anbindung gesteuert werden kann. Die Ergebnisse dieser Fallstudie wurden auf der CeBIT’98 in Hannover präsentiert.

7.3 Fertigungszelle

In einer weiteren Fallstudie wurde AutoFOCUS zur Entwicklung der Steuerung einer einfachen Fertigungszelle (siehe [19]) bestehend aus zwei Pressen und einem Bestückungsroboter eingesetzt. Hier wurde die SIMCENTER-Schnittstelle verwendet, um eine Tcl/Tk-basierte Visualisierungskomponente des “Forschungszentrum Informatik” aus Karlsruhe anzusteuern. Alternativ zur SIMCENTER-basierten Simulation wurde in dieser Studie aus der Spezifikation der Fertigungszelle über das in Abschnitt 5.4 beschriebene Exportinterface ein C-Programm generiert, das die Steuerung realisiert.

7.4 Tamagotchi

Im Rahmen einer vergleichenden Fallstudie wurde AutoFOCUS zusammen mit anderen Werkzeugen wie StateMate, ObjectGeode, ObjecTime und PEP zur Modellierung eines Tamagotchi verwendet ([17]). Hier zeigte sich, daß gerade der integrierte Ansatz von AutoFOCUS, insbesondere hinsichtlich der ausgewählten Beschreibungstechniken und deren Ausdrucksmöglichkeiten, trotz charakteristischer Mängel einer prototypischen Implementierung als sehr eingängig empfunden wird. Dies ermöglichte besonders eine einfache Einarbeitung in die eingesetzten Beschreibungstechniken. Darüberhinaus bestätigte sich auch die Bedeutung der Mehrbenutzerunterstützung sowie der Möglichkeit zur einfachen Generierung von Simulationen auf der Ebene der Beschreibungstechniken für den praktischen Einsatz.

8 Zusammenfassung und Ausblick

Im vorliegenden Artikel wurde ein Überblick über die grundlegenden Konzepte von AutoFOCUS und seinen aktuellen Entwicklungsstand gegeben. Die Durchgängigkeit des hier beschriebenen Konzepts und der Realisierung konnte bereits anhand verschiedener Fallstudien gezeigt werden.

Die Integration der Simulationskomponente SIMCENTER hat sich als methodisch wesentliche Unterstützung bei der Spezifikationsentwicklung erwiesen. Eine wesentliche Rolle spielt hierbei die Tatsache, daß die Generierung eines simulierbaren Prototypen in AutoFOCUS aus einer bereits erstellten Spezifikation automatisch möglich ist. Die Validierung einer Spezifikation wird dabei auf zwei unterschiedlichen Ebenen unterstützt: für den Spezifikationsersteller durch die Verwendung der gleichen Beschreibungsmittel für Spezifikation und Simulation und der damit verbundenen geringeren Fehleranfälligkeit, für den Anwendungsexperten durch

die Möglichkeit der Kopplung an eine anwendungsorientierte Visualisierung und der damit verbundenen Vermeidung von Verständigungsschwierigkeiten.

Da ein vollständig formaler Systementwurf aus technischer und ökonomischer Sicht unter den gegenwärtigen Voraussetzungen nicht realistisch ist, spielt auch weiterhin das Testen von entwickelten Systemen eine wesentliche Rolle im Systementwurf. Daher ist die Integration von Testmethoden in AutoFOCUS eines der aktuellen Arbeitsfelder; diese Arbeiten, zusammen mit Arbeiten zur Kombination von Entwurfs- und Beweiswerkzeugen, werden mit der Unterstützung des Bundesamtes für Sicherheit in der Informationstechnik BSI im Rahmen des Projekts QUEST bearbeitet.

Die starke Modularität der Beschreibungstechniken von AutoFOCUS bietet eine geeignete Grundlage für die Wiederverwendung von Spezifikationsanteilen als Spezifikationsbausteine. Obwohl noch nicht von der aktuellen Version unterstützt, sind die Voraussetzungen durch die der AutoFOCUS-Syntax und -Semantik zugrundeliegende konzeptuelle Basis bereits geschaffen (siehe Abschnitt 2.4). Aktuelle Arbeiten an AutoFOCUS beschäftigen sich daher mit der technischen Realisierung einer effizienten Nutzung von Spezifikationsbausteinen; der erste unternommene Schritt hierzu liegt in der Umstellung auf ein modellbasiertes Repository unter Verwendung einer objektorientierten Datenbank.

Danksagung Die beschriebenen Arbeiten entstammen einer Vielzahl von Projekten und entstanden durch die Beteiligung vieler Kolleginnen und Kollegen unserer Forschungseinrichtung. Ihnen, insbesondere den Mitgliedern aus den Projekten FOCUS und AutoFOCUS, sowie den Gutachtern für ihre hilfreichen Kommentare gilt daher unser besonderer Dank.

Literatur

1. Abrial, J.-R.: The B-Book: Assigning Programs to Meanings. Cambridge: Cambridge University Press 1996
2. Bechtel, R.: Einbettung des μ-Kalkül Model Checkers μ-cke in AUTOFOCUS. Diplomarbeit, Institut für Informatik, Technische Universität München 1999
3. Biere, A.: Effiziente Modellprüfung des μ-Kalküls mit binären Entscheidungdiagrammen. Dissertation, Universität Karlsruhe 1997
4. Bjørner, N., Browne, A., Chang, E., Colon, M., Kapur, A., Manna, Z., Sipma, H. B., Uribe, T. E.: STeP: The Stanford Temporal Prover (Educational Release) User's Manual. STAN-CS-TR 95-1562, Computer Science Department Stanford University 1995
5. Booch, G., Jacobson, I., Rumbaugh, J.: UML Summary. Cupertino, CA: Rational Software Corporation 1997
6. Broy, M., Dederichs, F., Dendorfer, C., Fuchs, M., Gritzner, T., Weber, R.: The Design of Distributed Systems - An Introduction to FOCUS. SFB-Bericht Nr.342/2-2/92 A, TUM-I 9202-2, Technische Universität München 1993
7. Burch, J. R., Clarke, E. M., McMillan, K. L., Dill, D. L., Hwang, L. J.: Symbolic Model Checking: 10^{20} States and Beyond. Information and Computation, 98(2):142–170 (1992)
8. Formula Software: FormulaGraphics Multimedia System. Siehe auch http://www.formulagraphics.com. 1998
9. Grosu, R., Klein, C., Rumpe, B., Broy, M.: State Transition Diagrams. Technischer Bericht TUM-I9630, Technische Univerität München 1996
10. Harel, D.: STATEMATE: A Working Environment for the Development of Complex Reactive Systems. IEEE Transactions on Software Engineering, 16(4):403–414 (1990)
11. Huber, F., Molterer, S., Schätz, B., Slotosch, O., Vilbig, A.: Traffic Lights - An AutoFocus Case Study. In: 1998 International Conference on Application of Concurrency to System Design, S. 282–294. Los Alamitos, CA: IEEE Computer Society 1998
12. Huber, F., Schätz, B.: Rapid Prototyping with AutoFocus. In: Wolisz, A., Schieferdecker, I., Rennoch, A. (Hrsg.), Formale Beschreibungstechniken für verteilte Systeme, GI/ITG Fachgespräch 1997, S. 343–352. St. Augustin: GMD Verlag 1997
13. Huber, F., Schätz, B., Einert, G.: Consistent Graphical Specification of Distributed Systems. In: Fitzgerald, J., Jones, C. B., Lucas, P. (Hrsg.), FME '97, LNCS 1313, S. 122–141. Berlin: Springer 1997
14. i-Logix: Rhapsody Reference Version 1.0. Andover, MA: i-Logix 1997
15. ITU-TS: Recommendation Z.120 : Message Sequence Chart (MSC). Genf: ITU 1996
16. Jones, M. P.: An Introduction to Gofer. Yale University, New Haven 1993
17. Kamsties, E., von Knethen, A., Philipps, J., Schätz, B.: Eine vergleichende Fallstudie mit CASE-Werkzeugen für formale und semiformale Beschreibungstechniken. In: Schätz, B., Spies, K. (Hrsg.), Formale Beschreibungstechniken FBT'99, GI/ITG Fachgespräch. München: Utz Verlag 1999
18. Lesny, C.: Sprachbeschreibung und Parserimplementierung der funktionalen Sprache "Frisco F". Diplomarbeit, Institut für Informatik, Technische Universität München 1997
19. Lötzbeyer, A.: Task Description of a Fault-Tolerant Production Cell. Available via http://www.fzi.de/divisions/prost/projects/korsys/korsys.html. 1996
20. Rational Software Corporation: Rational Rose 98 Product Overview. Siehe auch http://www.rational.com/products/rose. 1998
21. Schätz, B., Hußmann, H., Broy, M.: Graphical Development of Consistent System Specifications. In: Gaudel, M.-C., Woodcock, J. (Hrsg.), FME'96: Industrial Benefit and Advances In Formal Methods. Berlin: Springer 1996
22. Selic, B., Gullekson, G., Ward, P.: Real-Time Object-Oriented Modeling. New York, NY: John Wiley & Sons, Inc. 1994
23. Telelogic AB: SDT 3.1 Reference Manual. Malmö: Telelogic 1996
24. Wall, L., Schwartz, R. L.: Programming Perl. Sebastopol, CA: O'Reilly & Associates, Inc. 1992
25. Wilke Technology GmbH: BASIC Tiger. Siehe auch http://www.wilke.de/btiger.htm. 1999