KOMPONENTENBASIERTE ENTWICKLUNG VON AUTOMATISIERUNGSSYSTEMEN

P.Göhner
Institut für Automatisierungs- und Softwaretechnik (IAS), Universität Stuttgart
Pfaffenwaldring 47, D-70550 Stuttgart
P.Goehner@ias.uni-stuttgart.de

Zusammenfassung

Die Komplexität der Software in Automatisierungssystemen steigt immer mehr an. Die Erfahrung zeigt, daß die individuelle Softwareentwicklung, d.h. die spezifische Programmierung von Automatisierungssystemen auch bei Einsatz von Entwicklungsmethoden, Vorgehensmodellen und Softwarewerkzeugen in vielen Fällen die Forderungen hinsichtlich termingerechter Fertigstellung, Qualität und Kosten nur unzureichend erfüllt. Die aus den klassischen Ingenieurwissenschaften Übernommene komponentenbasierte Entwicklung wird heute mit mehr und mehr Erfolg in kommerziellen Applikationen eingesetzt. Um die komponentenbasierte Entwicklung auch für Automatisierungssysteme einsetzen zu können, müssen die Softwarekomponenten zusätzliche Anforderungen erfüllen. Ausgehend von einer allgemeinen Übersicht über die komponentenbasierte Entwicklung werden die spezifischen Anforderungen des Einsatzes für Automatisierungssysteme herausgearbeitet und ein Vorgehensmodell für die komponentenbasierte Entwicklung aufgestellt.

1 Probleme bei der klassischen Softwareentwicklung

Die Automatisierungstechnik ist eine Schlüsseltechnologie, die auch in Zukunft wesentlich die wirtschaftliche Entwicklung in unserem Land beeinflussen wird. Während bisher vor allem die Anlagenautomatisierung in der industriellen Produktion, im Verkehr, in der Energieversorgung und in der Medizin eine wichtige Rolle gespielt hat, hat in den letzten Jahren die Bedeutung der Produktautomatisierung bei industriellen Geräten, bei Haushalts- und Freizeitgeräten und im Kfz-Elektronikbereich sehr stark zugenommen. Gleichzeitig hat sich das Verhältnis zwischen Hardware- und Softwareanteilen drastisch in Richtung Software verschoben. So wird im Bericht der Zukunftskommission Wirtschaft 2000 festgestellt, daß *"die Software wettbewerbsentscheidend wird als wesentlicher Wertschöpfungsanteil und Kaufgrund von Produkten mit elektronischer Steuerung"*.

Zwar wurden in den letzten 20 Jahren erhebliche wirtschaftliche Anstrengungen unternommen, die Entwicklung von Softwaresystemen durch Vorgehensmodelle, Entwicklungsmethoden und Softwarewerkzeuge ähnlich wie in anderen Ingenieurdisziplinen zu unterstützen, um die Qualität, die Produktivität und die Wirtschaftlichkeit zu verbessern, aber in der Praxis zeigt sich, daß die erzielten Fortschritte auf dem Gebiet der Softwaretechnik und der industriellen und universitären Ausbildung nicht ausreichen, um das Problem der Softwareentwicklung in den Griff zu bekommen. Es gibt immer noch nicht "die`` Programmiersprache, Entwicklungsmethoden und Softwarewerkzeuge kommen und gehen und die Probleme haben sich aufgrund der gewachsenen Bedeutung von Software und der rasanten Fortschritte auf dem Hardwaresektor eher noch verschärft.

So wird in [1] festgestellt: *"Wir stehen im Treibsand. Die Computer werden immer schneller, billiger und kleiner. Doch der Nutzen ist zweifelhaft: Die Software hält nicht Schritt mit der Entwicklung. 75 Prozent aller komplexen Programmsysteme funktionieren nicht wie geplant.*

Die Anlaufprobleme auf dem Riesenflughafen von Denver sind ein Beispiel für die Softwarekrise."

Woher rühren diese Probleme? In den klassischen Ingenieurdisziplinen Elektrotechnik, Maschinenbau und Bauingenieurwesen bilden naturwissenschaftliche Erkenntnisse die Grundlage der Ingenieurtätigkeiten. Man bewegt sich in einer analogen Welt, in der kleine Änderungen in der Regel kleine Auswirkungen haben. Komplexe Systeme werden aus bewährten Komponenten zusammengesetzt.

Im Gegensatz dazu erscheint in der Ingenieurdisziplin Softwaretechnik das Denkbare machbar. Keine Naturgesetze behindern die Softwareentwicklung und die Grenzen wurden durch die Leistungsexplosion auf dem Computersektor nahezu aufgehoben. Man versucht, aus den Fehlern fehlgeschlagener Projekte zu lernen, wobei sich die nächsten Projekte aufgrund der neuen Anforderungen und der damit verbundenen größeren Komplexität wieder in neuen Dimensionen bewegen. In der digitalen Welt der Software haben bereits minimale Änderungen immense Auswirkungen, wie die Ariane-5 Katastrophe zeigt.

Die Hauptursachen für nicht termingerechte Fertigstellung, geringe Qualität, hohe Entwicklungs- und Wartungskosten liegen in der Komplexität von Softwaresystemen. Bei der klassischen Softwareentwicklung werden Softwaresysteme Top-Down so lange zerlegt, bis man sie in Form von Kleinstbausteinen, d.h. in den Anweisungen einer Programmiersprache, realisieren kann. Wie Bild 1 zeigt, nimmt mit jeder Zerlegungsebene die Komplexität des Gesamtsystems bezogen auf die verwendeten Softwareteile um den Faktor 10 zu.

F. Brooks [2] stellte schon 1987 fest, daß Komplexität eine grundlegende Eigenschaft von Software ist und keine zufällige. Die Realisierung geschieht in den meisten Fällen in Form von Anweisungen einer Programmiersprache. Ein Programm von 100.000 Quellzellen setzt sich bereits aus über 10.000 Bausteinen zusammen, wenn jeweils ein Baustein in 10 Unterbausteine zerlegt wird. Vergleicht man dazu die Vorgehensweise in anderen Ingenieurbereichen, stellt man fest, daß dort die früher verwendeten Kleinstbausteine in sehr vielen Bereichen durch vorgefertigte Komponenten ersetzt wurden, um die Komplexität der auch dort vorhandenen großen Systeme zu reduzieren. Kein Maschinenbauingenieur würde eine komplexe Maschine aus Kleinstbauteilen zusammensetzen, kein Bauingenieur würde ein Hochhaus aus Ziegelsteinen bauen, kein Elektroingenieur würde einen Fersehapparat mit Widerständen und Transistoren konstruieren.

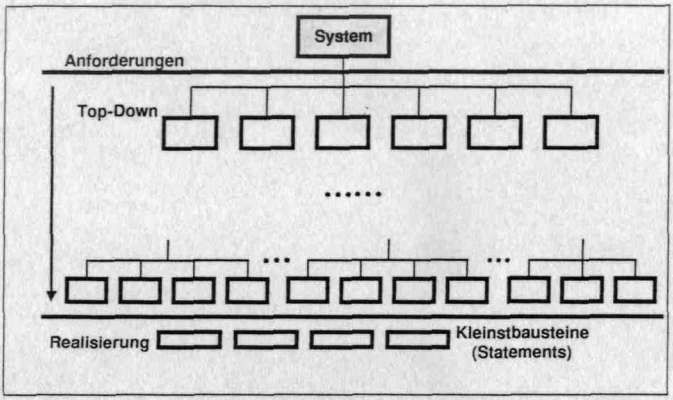

Bild 1: Zerlegung eines Softwaresystems in Einzelbausteien

Man erkennt in Bild 2, daß komplexe Systeme in den klassischen Ingenieurdisziplinen in ähnlicher Weise wie Softwaresysteme zerlegt werden, daß aber durch die Verwendung von Komponenten, die Produktcharakter haben und mit dem Ziel der Mehrfachverwendung entwickelt wurden, die Komplexität in der Zerlegungstiefe erheblich eingeschränkt wird.

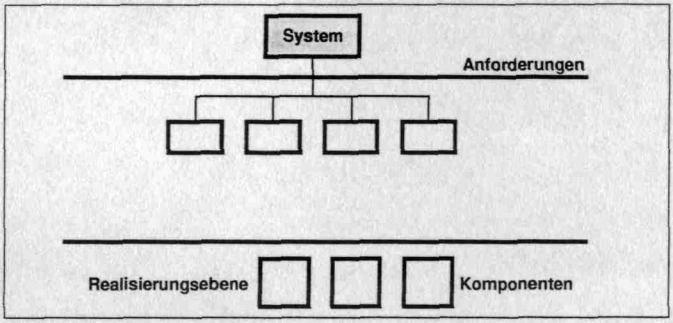

Bild 2: Zerlegung eines CAD-Systems in Einzelbausteine

Die Schlußfolgerungen, die man aus diesen Erkenntnissen ziehen kann, sind folgende:

- Die individuelle Neuentwicklung von Software ist äußerst aufwendig und wettbewerbsmäßig nicht mehr bezahlbar.
- Der Entwicklungsprozeß von Software muß umgestellt werden mit dem Ziel der Mehrfachverwendung von Softwarekomponenten.
- Softwarekomponenten werden als Mittelweg zwischen Individualentwicklung und Standardsoftware benötigt.

Durch den Einsatz vorhandener Komponenten ergeben sich Vorteile hinsichtlich der termingerechten Fertigstellung von Softwaresystemen, die Mehrfachverwendung dieser Komponen-

ten führt zu einer Erhöhung der Qualität und zu einer Reduzierung des Entwicklungs- und Wartungsaufwandes.

2 Komponentenbasierte Softwareentwicklung

Der ökonomische Anstoß für die Wiederverwendung von Software ist derselbe wie der für die Verwendung von genormten Teilen im Auto- oder Flugzeugdesign. Normteile sind billiger, zuverlässiger und in der Regel leichter zu reparieren oder zu ersetzen, wenn Fehler auftreten. Die zentrale Leitidee lautet „buy, don't build", d.h. wie in Bild 3 skizziert, werden komplexe Systeme aus Softwarekomponenten mittels Auswahl, Konfigurierung und Verknüpfung zu einer Applikation zusammengebaut.

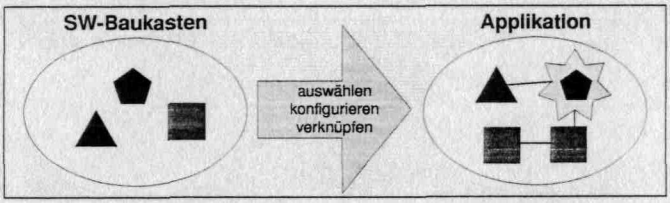

Bild 3: Komposition von Softwaresystemen

Unter einer Softwarekomponente versteht man dabei nicht zwangsläufig ausführbaren Code.[3] Eine Softwarekomponente kann auch eine Beschreibung sein, aus der Code generiert werden kann, bzw. jedes wesentliche Ergebnis des Software-Lifecycles, wie z.B. eine Requirement-Spezifikation, Testfälle usw. [4]

Wie in Bild 4 dargestellt, ist die Idee der Softwarewiederverwendung nicht neu, während aber früher bei Funktionen, Modulen und auch bei Klassen die Implementierungssicht im Vordergrund stand, orientieren sich heute Komponenten stärker an den zu realisierenden Applikationen.

Um Softwaresysteme durch Konstruktion aus Komponenten aufbauen zu können, benötigen Softwarekomponenten eine Reihe von Eigenschaften. Eine Softwarekomponente muß eine in sich abgeschlossene funktionale Einheit mit vollständig spezifizierten Schnittstellen bilden, sie muß gut dokumentiert, qualitativ hochwertig und unverändert mehrfach verwendbar sein. Zusätzlich soll eine Softwarekomponente unabhängig von anderen Softwarekomponenten sein, an eine gegebene Aufgabenstellung anpassbar und bezüglich bestimmter Eigenschaften konfigurierbar sein.[5].

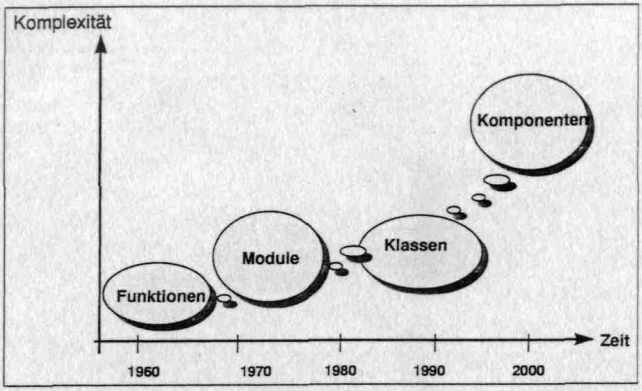

Bild 4: Softwarewiederverwendung

3. Softwarekomponenten in Automatisierungssystemen

Im Gegensatz zu kommerziellen Anwendungen, wo die Verwendung von Softwarekomponenten in vielen Bereichen bereits begonnen hat [6], bereitet der Einsatz in Automatisierungssystemen Schwierigkeiten. Automatisierungssysteme sind Echtzeitsysteme, bei denen neben den funktionalen Anforderungen, zeitliche Anforderungen, Anforderungen an die Zuverlässigkeit und die Sicherheit realisiert werden müssen, wobei zusätzlich ein hoher Kostendruck vor allem bei der Produktautomatisierung auftritt. In Bild 5 sind die 3 Dimensionen der Komplexität von Automatisierungssystemen dargestellt.

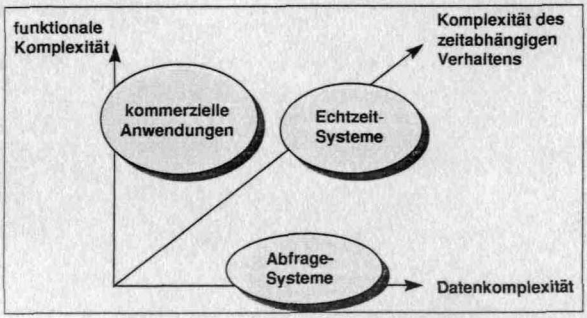

Bild 5: Dimensionen der Komplexität von Automatisierungssystemen

Aufgrund dieser Anforderungen sind die Komponentenkonzepte aus der kommerziellen Datenverarbeitung nicht direkt auf Automatisierungssysteme anwendbar. Man benötigt Systemarchitekturen, die es ermöglichen, Softwarekomponenten, die bestimmte zeitliche Anforderungen erfüllen, zu Gesamtsystemen zu integrieren, ohne die Zeiteigenschaften zu beeinträchtigen. Neben der funktionalen Kapselung von Komponenten spielt deshalb auch die zeitliche Kapselung von Komponenten eine wichtige Rolle. Die Gewährleistung des Zeitverhaltens läßt sich nur dann erreichen, wenn einerseits die Zeiteigenschaften einzelner Komponenten ein-

deutig festgelegt sind, und wenn andererseits bei der Komposition von Komponenten nur Mechanismen angewendet werden, die das Zeitverhalten des so erzeugten Gesamtsystems bestimmbar machen.

Die heute üblicherweise beim Entwurf von Echtzeitsystemen eingesetzten ereignisgesteuerten Architekturen sind bezüglich des Zeitverhaltens nicht deterministisch, d.h. es lassen sich damit keine harten Zeitanforderungen realisieren. Zeitgesteuerte Systeme basieren auf strengeren Annahmen über die Regelmäßigkeit von Abläufen und sind daher weniger flexibel, aber einfacher zu analysieren und damit zu berechnen.

Eine Möglichkeit, zeitgesteuerte Echtzeitsysteme aus Softwarekomponenten aufzubauen, ist der Einsatz sog. synchroner Softwarekomponenten. Synchrone Softwarekomponenten beruhen auf endlichen Automaten und einem synchronen Zeitmodell. Sie sind durch einen gemeinsamen Takt perfekt synchronisiert (zeitgesteuert), haben ein deterministisches Verhalten und besitzen nur eine implementierungsabhängige charakteristische Zeitkonstante. Sie kommunizieren durch Signal-Broadcasting und laufen parallel zu anderen Komponenten. Durch die Verwendung zeitgesteuerter Kommunikationsprotokolle mit integrierter Uhrsynchronisation kann der synchrone Ansatz auch in verteilten Systemen, wie sie in der Automatisierungstechnik sehr häufig vorkommen, vorteilhaft realisiert werden.[7]

Der Entwickler von Softwarekomponenten für Automatisierungssysteme steht vor einem Dilemma. Auf der einen Seite weiß er, daß die Herstellung von Softwarekomponenten einen Mehraufwand bedeutet, der sich nur dann lohnt, wenn die Komponenten wiederverwendet werden können. Für den Nutzer müssen sie deshalb eine ausreichende Attraktivität ihrer Anwendung offerieren, d.h. eine interessante Funktionalität mit einer Reihe von Freiheitsgraden. Auf der anderen Seite ist vor allem aus der Produktautomatisierung bekannt, daß aufgrund der großen Stückzahl äußerst sparsam mit Speicherplatz und Laufzeit umgegangen werden muß. Konkrete Softwarekomponenten, d.h. Softwarekomponenten in Codeform haben hier den Nachteil, daß eine mächtige Funktionalität mit Freiheitsgraden gleichzeitig auch entsprechende Ressourcen zur Laufzeit benötigt. Aus diesem Grund reichen konkrete binäre Komponenten nicht aus. Man benötigt generische Komponenten auf Modellebene, die durch Parametrierung und Konfigurierung an die gegebene Aufgabenstellung angepaßt werden können.

Bei einer generischen Komponente handelt es sich um einen wiederverwendbaren Systembaustein mit einer abgeschlossenen funktionalen Einheit, die einen bestimmten Problembereich repräsentiert. Über eine Konfigurierung der Systemeigenschaften und Parametrierung kann eine derartige Komponente an spezielle Anforderungen und Randbedingungen einer zu realisierenden Applikation angepaßt werden. Das Entscheidende ist hierbei, daß die Anpassungen auf Modellebene der generischen Komponenten erfolgt, auf einer Ebene, auf der Anpassungsmöglichkeiten wesentlich flexibler und effizienter durchzuführen sind. Erst nach erfolgter Anpassung wird für ein Zielsystem automatisch Code generiert. Der Weg einer generischen Komponente zur konkreten Komponente führt also im wesentlichen über eine Instanziierung in Form einer Anpassung durch Konfigurierung und Parametrierung und anschließender Code-Generierung. Auf diese Weise lassen sich die notwendigen funktionalen Anforderungen erfüllen, bei gleichzeitiger Code-Effizienz, d.h. es gibt keine Elemente, die nicht zur Problemlösung beitragen. Bei der Konfigurierung und Parametrierung zielen mögliche Einstellungen nicht immer in dieselbe Richtung. So kann beispielsweise eine gewählte Eigen-

schaft zur Fehlertoleranz der Vorgabe der Einhaltung bestimmter Zeitschranken widersprechen. Um ein Fehlverhalten zur Laufzeit ausschließen zu können, ist es demnach erforderlich, bereits während des Anpassungsprozesses eine Plausibilitätsprüfung der vorgenommenen Einstellungen durchzuführen. Somit können dem Nutzer etwaige Diskrepanzen zwischen eingestellter Soll- und zur Laufzeit tatsächlich vorhandenen Ist-Ausprägung sowie widersprüchliche Einstellungen angezeigt werden.

In Bild 6 ist ein Vorgehensmodell für die komponentenbasierte Entwicklung von Automatisierungssystemen dargestellt.

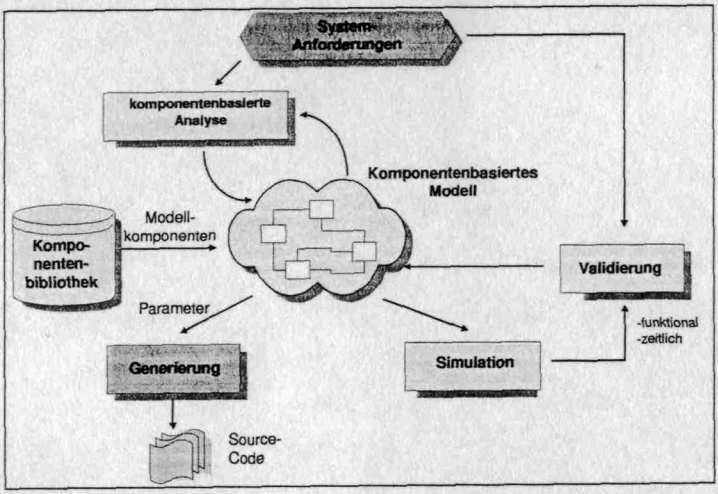

Bild 6: Vorgehensmodell für die komponentenbasierte Entwicklung von Automatisierungssystemen

Ausgehend von den Systemanforderungen wird eine komponentenbasierte Analyse durchgeführt, bei der unter Einsatz von generischen Komponenten aus einer Komponentenbibliothek ein komponentenbasiertes Modell aufgebaut wird. Das komponentenbasierte Modell beinhaltet ein funktionales Modell, ein dynamisches Modell, ein Architekturmodell und ein Modell des technischen Prozesses. Durch Simulation wird das komponentenbasierte Modell auf funktionale und zeitliche Eigenschaften überprüft. Sind die Systemanforderungen erfüllt, werden die generischen Komponenten instantiiert.[8]

Eine besondere Form der komponentenbasierten Entwicklung ist der Einsatz von Frameworks für ausgewählte Anwendungsbereiche der Automatisierungstechnik. Durch Frameworks werden viele Entwicklungsentscheidungen bereits vorweggenommen, und die Möglichkeit, als Entwickler Einfluß zu nehmen und damit Fehler zu machen, ist reduziert.[9]

Im Gegensatz zur allgemeinen komponentenbasierten Entwicklung, in der unabhängige Komponenten wiederverwendet werden, sind bei Frameworks die einzelnen Komponenten nicht unabhängig voneinander, sondern Bestandteile einer fest vorgegebenen übergeordneten Architektur, die für diesen speziellen Anwendungsbereich zugeschnitten ist. Frameworks bieten

sich vor allem dort an, wo Produkt- oder Anlagenautomatisierungssysteme in Form vieler Varianten zum Einsatz kommen. Statt für jede Variante neue Software zu entwickeln bzw. aus Komponenten neu aufzubauen, bietet es sich hier an, auch zu Lasten einer breiten Einsetzbarkeit, nur die Teile flexibel zu gestalten, bei denen sich die Varianten unterscheiden.

4 Zusammenfassung und Ausblick

Die individuelle Programmierung von Automatisierungssystemen, sei es in der Anlagenautomatisierung oder in der Produktautomatisierung, ist nicht geeignet, um die zunehmende Komplexität dieser Softwaresysteme zu bewältigen. Die komponentenbasierte Entwicklung, die auf ähnlichen erfolgreichen Vorgehensweisen anderer Ingenieurdisziplinen beruht, bietet die Möglichkeit, qualitativ hochwertige Systeme termingerecht zu angemessenen Entwicklungs- und Wartungskosten zu realisieren.

Im Vergleich zu kommerziellen Applikationen treten bei der komponentenbasierten Entwicklung von Automatisierungssystemen aufgrund der Echtzeiteigenschaften und der Stückkostenproblematik zusätzliche Anforderungen auf, die dazu führen, daß Softwarekomponenten sowohl zeitlich als auch funktional gekapselt werden müssen, beziehungsweise daß die Konfigurierung von Komponenten, d.h. die Anpassung an die jeweilige Aufgabenstellung, auf Modellebene stattfinden muß.

Die Forschung auf dem Gebiet ***komponentenbasierte Entwicklung von Automatisierungssystemen*** befindet sich derzeit im Aufbruch. Es gibt zwar für ausgewählte Gebiete in der Leittechnik, bei der Steuerung von Fertigungsanlagen und bei der Steuerung von Werkzeugmaschinen erste Ansätze - eine komponentenbasierte Entwicklungsmethode und Entwicklungswerkzeuge für die Komposition von Komponenten, vergleichbar mit CAD-Werkzeugen, sind derzeit noch nicht verfügbar.

Die Forschungsarbeiten am IAS sind im Forschungsschwerpunkt CASCADE (**C**omputer **A**ided **S**oftware **C**omposition for **A**utomation **S**ystem **D**evelopment and **E**valuation) zusammengefaßt und beschäftigen sich sowohl mit der Bereitstellung von Komponenten für Automatisierungssysteme als auch mit der Integration von Komponenten zum Aufbau von Automatisierungssystemen.

Literaturverzeichnis

[1] Spiegel 4/1995: Seite 186

[2] Brooks, F.: Vom Mythos des Mann-Monats. 1. Aufl. Bonn Addison-Wesley 1987. The mythical man-month: essays on software engineering. Anniversary ed. Reading, Mass: Addison-Wesley Publ. 1995

[3] Eisenecker, U.: Generative Programmierung und Komponenten, OBJEKTspektrum 3/97, S. 80-84

[4] Brown, A.W.; Wallnau, K.C.: Engineering of Component-Based Systems, Selected Papers from the Software Engineering Institute, Carnegie Mellon University, Pittburgh, IEEE Computer Society, 1996, S. 7-15

[5] Szyperski, C.: Component Software. Addison-Wesley, 1997

[6] Heuser-Hasenpatt, H. et al.: Bausteinorientierte Anwendungsentwicklung, OBJEKTspektrum 3/97, S. 40-51

[7] Kopetz, H.; Grünsteidl, G.: TTP-A Protocol for Fault-Tolerant Real-Time Systems, IEEE Computer, Vol. 27 NO. 1, Jan. 1994, IEEE Computer Society, 1994

[8] Fleisch, W.; Göhner, P.; Belschner R.; Gunzert, M.: Building Complex Automation Systems Based on the Simulation of Object-Oriented Basic Components, Proc European Simulation Multi-Conference, Budapest, Hungary, 1996

[9] Berg, K.; Marek, B.: Ausprägung von Componentware, Beherrschung von Informationssystemen: Technische Beiträge und Praxisprogramm. Heinrich C. Mayr (Hrsg.): Proceedings-Reihe der Informatik '96, Band 0, Klagenfurt, 1996, S. 37-49