

Effiziente Entwicklung von Prozeßautomatisierungssoftware

Wolfgang Narzt, Josef Pichler, Klaus Pirklbauer, Gustav Pomberger, Martin Zwinz¹

Prozeßautomatisierungssoftware ist ein typisches Beispiel für Individualsoftware, für die in der Regel große Teile für jede Anlage neu erstellt werden. Dieser Aufsatz zeigt am Beispiel des Warmwalzbereichs, wie eine entsprechende Softwarearchitektur dazu beitragen kann, die Entwicklungszeit zu verkürzen und die Kosten zu senken und dabei gleichzeitig die Qualität, insbesondere die Anpaßbarkeit und die Wartbarkeit zu erhöhen.

Schlüsselwörter: Prozeßautomatisierungssoftware, Warmwalzwerk, komponentenorientierte Software, Application Framework, CORBA

Efficient Development of Process Automation Software. Process automation software is a typical example of software with great parts developed individually for each plant. This paper shows a software architecture for the hot rolling mill domain which helps to shorten development time and to reduce costs and to simultaneously increase quality, especially adaptability and maintainability.

Keywords: process automation software, hot rolling mill, component-oriented software, CORBA

Motivation und Überblick

Jeder, der Software einsetzt, erwartet sich ein zuverlässiges Produkt, das im Bedarfsfall an geänderte Bedingungen leicht anpaßt werden kann, und das nicht zu teuer ist. Gerade diesen Anforderungen kommt im Automatisierungsbereich eine besondere Bedeutung zu. Anlagen müssen einerseits zuverlässig automatisiert werden, um materielle und auch personelle Schäden zu vermeiden und um eine effiziente Maschinennutzung zu ermöglichen. Andererseits ist eine einfache Änder- und Anpaßbarkeit der Software notwendig, um auf Änderungen in der Struktur und in der Betriebsweise der Anlage rasch reagieren zu können. Nicht zuletzt ist auch der Preis ein wichtiger Punkt, da Automatisierungssoftware meist Individualsoftware ist, deren Preis nicht durch hohe Verkaufszahlen gesenkt werden kann.

Die Minimierung des Implementierungsaufwands durch den Einsatz vorgefertigter Softwarebausteine kann eine Erhöhung der Qualität bei gleichzeitiger Verminderung der Entwicklungszeit bewirken. Die Verminderung der Entwicklungszeit führt dann zu einer Verringerung der Entwicklungskosten, wenn die Entwicklungszeitverkürzung nicht durch den Einsatz von mehr Personal, sondern durch die Wiederverwendung und vereinfachte Anpaßbarkeit von Softwarearchitekturbausteinen erreicht wird. Allerdings verursacht die Entwicklung der Softwarearchitekturbausteine auch Kosten, die sich in der Regel erst nach mehrmaliger Wiederverwendung des Komponentenvorrates amortisieren.

Dieser Aufsatz zeigt am Beispiel eines Prozeßautomatisierungssystems für Warmwalzwerke in der Stahlindustrie, wie eine entsprechende Softwarearchitektur dazu beitragen kann, daß

¹ Dipl.-Ing. Wolfgang Narzt, Institut für Praktische Informatik, Software; Dipl.-Ing. (FH) Josef Pichler, Dr. Klaus Pirklbauer, Prof. Dr. Gustav Pomberger, Dipl.-Ing. (FH) Martin Zwinz, Institut für Wirtschaftsinformatik, Software Engineering, Johannes Kepler Universität Linz, Altenbergerstraße 69, A-4040 Linz

Automatisierungssoftware die oben erwähnten Anforderungen erfüllt. Insbesondere wird darauf eingegangen, welche Arten von Softwarebausteinen (z.B. objektorientierte Softwarebausteine, Bausteine basierend Frameworks) und welche Eigenschaften der Softwarebausteine Vorteile für die Wartbarkeit, Anpaßbarkeit und Wiederverwendbarkeit bewirken.

Prozeßautomatisierungssoftware für Warmwalzwerke eignet sich besonders für die Untersuchung der Wiederverwendbarkeit bestimmter Softwarearchitekturen und des daraus resultierenden Einsparungspotentials, weil in diesem Bereich Automatisierungssysteme für jeden Kunden individuell erstellt werden. Daß dabei häufig der Großteil der Komponenten jedesmal neu entwickelt wird, obwohl man Teile für mehrere Anlagen verwenden könnte, verursacht einen hohen Teil der Entwicklungskosten. Auch die Qualität wird durch wiederholte Neuentwicklung nicht nur positiv beeinflusst, weil dadurch auch stets neue Fehler und Mängel dazukommen können. Außerdem sind die bestehenden Systeme meist so wenig flexibel, daß beispielsweise der Einbau eines zusätzlichen Meßgeräts in der Walzstraße unnötig hohe Anpassungsarbeiten an der Software verursacht.

Die hier vorgestellte Architektur wurde im Rahmen des ESPRIT-Projekts Nr. 22897 „REFORM“ [1] entwickelt. In diesem Projekt arbeiten sechs europäische Partner aus Österreich, Deutschland und Schweden an der Implementierung einer flexiblen Softwarearchitektur für Warmwalzwerke.

Anwendungsgebiet

Das Prozeßautomatisierungssystem für Warmwalzwerke ist Teil eines umfassenden Automatisierungssystems, das grundsätzlich in drei Ebenen [2] eingeteilt werden kann.

- Die *Ebene 1*, sie wird auch als *Basisautomatisierungsebene* bezeichnet. Meßgeräte, Sensoren und andere Aggregate dieser Ebene liefern Informationen über die Anlage und die gewalzten Bänder an die Ebene 2.
- Die *Ebene 2*, *Prozeßautomatisierungsebene* genannt, empfängt die Informationen von

Ebene 1 und von vor- oder nachgeschalteten Anlagenteilen und berechnet zur Steuerung notwendige Daten, die über die Basisautomatisierung an die Walzgerüste, die Antriebe und die sonstigen Aggregate weitergegeben werden. Außerdem werden gemessene und berechnete Daten für Auswertungen und zu Kontrollzwecken zur Verfügung gestellt.

- Die *Ebene 3*, die sogenannte *Produktionsplanungsebene*, führt die Auslastungs- und Auftragsplanung (Walzprogramm) durch. Um auf auftretende Probleme reagieren zu können, muß die Ebene 2 Zustandsdaten an die Ebene 3 weitergeben.

Dieser Aufsatz beschreibt das Prozeßautomatisierungssystem auf Ebene 2, das die Steuerung vom Eintreffen einer Bramme (Stahlblock) beim Stoßofen, über die Erhitzung, die Grobwalzung in der Vorstraße, die Feinwalzung in der Fertigstraße, die Kühlung in der Kühlstrecke, bis zur Aufwicklung des Stahlbandes bei der Haspel übernimmt.

Die Kernfunktionen des Prozeßautomatisierungssystems auf Ebene 2 sind Materialverfolgung und Meßwertverarbeitung auf der einen Seite und Berechnungen der Steuerungsdaten (Modellberechnungen genannt) für den Walzprozeß auf der anderen Seite (siehe Abb. 1).

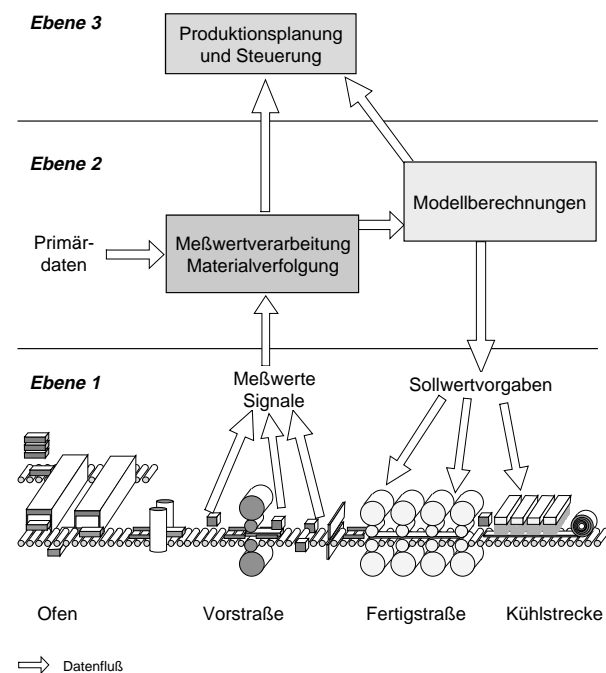


Abb. 1: Struktur eines Automatisierungssystems

Aufgabe der *Materialverfolgung* ist es, die Bewegung der Bänder zu verfolgen und an bestimmten Punkten Ereignisse auszulösen. Die *Meßwertverarbeitung* hingegen sammelt und speichert alle notwendigen Daten und gibt sie in statistisch aufbereiteter Form an die mathematischen Modelle weiter, deren Berechnungsergebnisse dazu verwendet werden, den Produktionsprozeß zu steuern und zu optimieren. Die meisten dieser Berechnungen werden vor, während und nach den einzelnen Walzvorgängen angestoßen.

Abbildung 1 zeigt einen zyklischen Datenfluß beginnend in Ebene 1. Alle benötigten Daten (Signale, Meßwerte, Einstellungen, usw.) werden in Form von Telegrammen (in Abständen von ca. 100 ms) an die Kernfunktionen Materialverfolgung und Meßwertverarbeitung weitergeleitet, wo sie für die mathematischen Modelle zur Verfügung stehen. Die berechneten Sollwerte gehen schließlich wieder zur Basis zurück und beeinflussen so den Walzprozeß. Damit ist der Kreis geschlossen. Das Produktions- und Planungssystem wird laufend über die Vorgänge in der Anlage informiert und berechnet, wenn nötig, eine neue Walzfolge, um den Produktionsprozeß zu optimieren.

Zusätzlich zu den Kernfunktionen gibt es noch weitere Prozesse, die für ein vollständig funktionsfähiges Automatisierungssystem notwendig sind: Die sogenannte Primärdatenverwaltung versorgt das Kernsystem mit den Maßen und Zielwerten der zu walzenden Brammen. Eine Benutzerschnittstelle (MMI — Man Machine Interface) ermöglicht es dem Operator, den Walzvorgang zu überwachen und steuernd einzugreifen. Mittels Protokollmechanismus können für jedes Band Berichte über den Produktionsprozeß erzeugt werden. Gegebenenfalls spielen noch weitere Prozesse, wie z.B. ein Logging-Mechanismus, eine Rolle im Automatisierungssystem.

Prozeßautomatisierungssoftwarearchitektur

Die im REFORM-Projekt entwickelte Softwarearchitektur für den Warmwalzwerksbereich zeigt Abbildung 2 schematisch.

Das Kernsystem der Ebene 2, bestehend aus Materialverfolgung und Meßwertverarbeitung, empfängt die Daten von der Basisautomatisierung in Form von Telegrammen. Meist sind die Kommunikationsprotokolle in Ebene 1 unterschiedlich zu denen in Ebene 2, daher wurde beim Design der Architektur dafür gesorgt, daß ein Telegrammverarbeitungsprozeß die Telegrammkommunikation (Ebene 1) in Methodenaufrufe via CORBA [3] (Ebene 2) umwandelt. Diese Zwischenschicht ermöglicht es auch, Basisautomatisierungssoftware beliebiger Hersteller zu verwenden. Verschiedene mathematische Modelle (M1, M2, M3), die Benutzerschnittstelle (MMI), die Primärdatenverwaltung und der Protokollmechanismus sind, wie in Abbildung 2 gezeigt, mit dem Kernsystem verbunden und bilden so das Prozeßautomatisierungssystem.

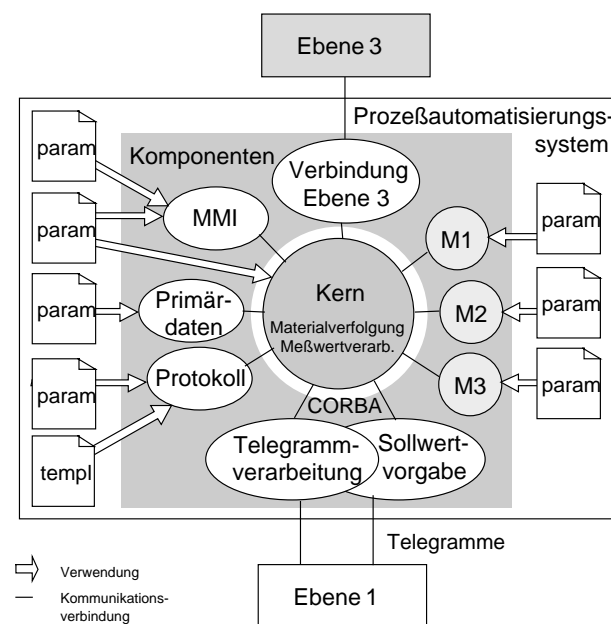


Abb. 2: Softwarearchitektur

Es ist heute selbstverständlich, daß Programme objektorientiert unter Heranziehung von Design-Pattern (Entwurfsmustern) [4,5] erstellt werden. Auch die Teile des hier vorgestellten Automatisierungssystems von Warmwalzwerken sind auf diese Weise entwickelt und implementiert worden. Beispielsweise wurden sämtliche Aggregate (Meßgeräte, Sensoren, Walzgerüste usw.), die in einer Anlage vorkommen können, mit entsprechenden Aggregat-Klassen implementiert. Sie können durch das Composite-Design-Pattern [4] hierarchisch

angeordnet werden und die von diesen Klassen gebildeten Objekte speichern aggregatbezogene Daten, die sie von der Basisautomatisierung erhalten. Ein Material-Objekt (für die Bramme und später das Band) verwaltet die verschiedensten bandbezogenen Daten. Der Vorteil ist, daß die Datenhaltung für die Aggregate und für das Band dadurch nicht auf bestimmte Datenelemente mit festgelegten Typen beschränkt ist. Eine gemeinsame Basisklasse für Objekte zur Speicherung von Werten und für Objekte zur Speicherung standardisierter Wertnamen zusammen mit entsprechenden Maßeinheiten sorgen für leichte Erweiterbarkeit (ohne Änderung im Quelltext) und dafür, daß die Modellberechnungen nicht explizit angepaßt werden müssen.

Objektorientierung alleine reicht aber noch nicht aus, um die erforderliche Flexibilität einer Basisarchitektur für ein flexibles Softwaresystem in befriedigendem Ausmaß zu gewährleisten. Man kann zwar durch Vererbung vermeiden, immer wieder ähnliche Softwareteile entwickeln zu müssen; und man kann durch Polymorphismus und dynamische Bindung die Anpassungsfähigkeit erhöhen, dennoch gehört zu einem flexiblen System noch mehr:

Jedes Warmwalzwerk besteht aus Produktionsabschnitten, die in sehr ähnlicher Weise funktionieren. Jedes Warmwalzwerk ist aus Aggregaten aufgebaut (Sensoren, Meßgeräte, Rollgänge, Walzgerüste, usw.) — in manchen Anlagen gibt es mehr, in manchen weniger. Außerdem finden wir die drei Automatisierungsebenen in fast allen Anlagen wieder. Es ist daher naheliegend, die Grundfunktionen eines Warmwalzwerks durch Rahmenprogramme (Application Frameworks) bereits vorzugeben, d.h. eine Application Framework-basierte Architektur für die Prozeßautomatisierung zu wählen.

Ein Application Framework ist eine besonders auf Wiederverwendbarkeit getrimmte Sammlung von Klassen für ein bestimmtes Anwendungsgebiet, ein Gerüst, das bereits einen Bauplan zur Lösung einer spezifischen Problemklasse bereitstellt [6]. Die Klassen für das Prozeßautomatisierungssystem im REFORM-

Projekt bilden ein Application Framework, das die Grundfunktionen und den Steuerfluß eines Automatisierungssystems enthält. Dieses Framework ist so gestaltet, daß es eine generische Lösung darstellt, die an bestimmte Anlagentypologien durch Ableitung oder Parametrierung angepaßt werden kann.

Das Framework stellt also die Grundfunktionen eines Prozeßautomatisierungssystems gewissermaßen als Halbfabrikat zur Verfügung. Beispielsweise ist der gesamte Ablauf der Materialverfolgung im Framework bereits eingebaut, weil sich dieser in der Regel nicht ändert. Das Framework stellt auch die wesentlichen Funktionen der Meßwertverarbeitung zur Verfügung. Es bietet spezielle Speicherklassen an, von denen Objekte gebildet werden können, die während des Walzprozesses alle relevanten Daten (Meßwerte, berechnete Werte oder Vorgabewerte) speichern bzw. statistisch aufbereiten. In jedem Warmwalzwerk können somit unterschiedliche Daten gehalten werden, aber die Art der Datenspeicherung ist bereits durch das Framework vorgegeben.

Mit der Strategie, so viele Algorithmen wie möglich (in verallgemeinerter Form) in das Framework aufzunehmen, kann die Entwicklungszeit erheblich gekürzt werden. Allerdings läuft man dadurch Gefahr, an Flexibilität einzubüßen. Beispielsweise könnte auch die Basisautomatisierung die Materialverfolgung übernehmen und anstatt Sensordaten Bandpositionen übermitteln. Derartige Veränderungen in der Art der Daten können das Framework unbrauchbar machen, wenn es nicht von vornherein flexibel genug ist, um für solche Veränderungen konfiguriert zu werden. Das REFORM-Framework erfüllt diese Anforderungen.

Softwarekomponenten als Mittel zur Erhöhung der Flexibilität

Obwohl das oben skizzierte Framework ganz wesentlich zur Vereinfachung des Software-Entwicklungsprozesses durch Erhöhung der Wiederverwendbarkeit beiträgt, muß auch ein Austausch oder ein Hinzufügen von Teilen der Automatisierungssoftware einfach und schnell möglich sein. Zum Beispiel soll eine neue

Auswertung oder ein neues Modell einfach mit dem laufenden Kernsystem kombiniert werden können. Dazu ist es notwendig, abgeschlossene Teile der Software mit entsprechend flexibel gehaltenen Schnittstellen als Softwarekomponenten zu entwickeln.

Das Ziel bei komponentenorientierter Softwareentwicklung [siehe dazu z.B. 7,8,9] ist, ein System aus voneinander unabhängigen, jedoch verbindbaren Teilen zu bauen. Bei steigender Komponentenanzahl steigt die Verbindungskomplexität überproportional. Im Mittelpunkt der Konstruktion von Softwaresystemen steht somit das Verbinden einzelner Komponenten über flexible Schnittstellen. Die Flexibilität wird durch Reduktion der Schnittstellen auf einen generischen Event-Mechanismus erreicht.

Für die Prozeßautomatisierung müssen die Komponenten in der Regel mit großen Datenmengen versorgt werden. Zum Beispiel benötigen die Modelle Daten aus dem Walzprozeß, die jedoch von Anlage zu Anlage verschieden sind. Es wäre zu mühsam und zu aufwendig, müßte man die Datenversorgung jedesmal neu implementieren. Man benötigt daher einen Mechanismus, mit dessen Hilfe die Versorgung einer Komponente mit den gewünschten Daten einfach und schnell bewerkstelligt werden kann.

Das Design der Softwarekomponenten muß aus wirtschaftlicher und technischer Perspektive so gestaltet sein, daß es möglich ist, daß auch bereits existierende Softwarekomponenten (sogen. legacy software) wiederverwendet, d.h. integriert werden können. Zum Beispiel arbeitet jedes Warmwalzwerk mit eigenen mathematischen Modellen, die teilweise über viele Jahre hinweg entwickelt worden sind. Auch solche älteren Modellimplementierungen müssen in neuen Projekten verwendbar bleiben.

In dem hier vorgestellten Ansatz werden für die Herstellung von Prozeßautomatisierungssoftware für Walzwerke komponentenorientierte Techniken verwendet, um möglichst hohe Flexibilität zur Anpassung von Frameworks an geänderte Anforderungen bei neuen Walzwerkprojekten zu erreichen.

Die Teile rund um den Kern des Frameworks (z.B. Telegrammverarbeitung, siehe Abb. 2) werden als parametrisierbare Komponenten realisiert, denen eigenständige Prozesse zugeordnet sind. Diese kommunizieren mit dem Kern über einen ereignisbasierten Mechanismus, der in Form einer vordefinierten CORBA-Schnittstelle realisiert ist.

Mathematische Modelle sind als eigenständige Komponenten realisiert, da sie sich von Walzwerk zu Walzwerk unterscheiden und daher leicht austauschbar sein müssen. Auch Teile, wie z.B. der Protokollmechanismus das MMI, sind als Komponenten ausgeführt, die mit dem Kern über den selben ereignisgesteuerten Mechanismus kommunizieren. Im Ergebnis erreicht man damit eine einheitliche Softwarestruktur im gesamten Prozeßautomatisierungssystem. Damit wird nicht nur die Anpaßbarkeit und Erweiterbarkeit verbessert, sondern auch die Strukturkomplexität verringert.

Eine wesentliche Designentscheidung war, die in Abbildung 2 angeführten Komponenten als grob granulare Komponenten zu gestalten, d.h. es wurde viel Funktionalität in die einzelnen Komponenten gepackt. Diese Designentscheidung gründet sich primär auf domänenspezifische Gegebenheiten; eine feingranularere Architektur würde in diesem spezifischen Anwendungsbereich die Komplexität der Wechselwirkungen zwischen den Komponenten erhöhen, ohne daß dadurch die Anpaßbarkeit der Architektur verbessert wird. Außerdem würde die Integration „alter“ Softwarebausteine erschwert, weil die üblicherweise vorhandenen Softwarebausteine (z.B. mathematische Modelle) ebenfalls grob granular sind. Besonders wichtig ist es, die Komponenten so zu gestalten, daß die Kopplung von Komponenten (d.h. bestehende Komponenten mit neu entwickelten oder zugekauften Komponenten) auch von Programmierern, die mit komponentenorientierten Techniken nicht vertraut sind, auf einfache Weise bewerkstelligt werden kann. In dem hier vorliegenden Ansatz wird dazu das Konzept der Konfigurierbarkeit verwendet.

Konfigurierbarkeit

Eine einfache Kopplung von Komponenten ist aber nur dann möglich, wenn 1. die Komponenten exakt die geforderte Funktionalität bereitstellen und 2. die Kopplung keine speziellen Programmierungstechniken erfordert. Da sich die einzelnen Anlagenautomatisierungen aber nahezu immer in Details unterscheiden, ist eine Konfigurierbarkeit mittels Konfigurationsdateien ein vernünftiger Ansatz dafür. Diese Konfigurationsdateien (ASCII-Dateien) sind so gestaltet, daß mit ihnen nicht nur einfache Parameterwerte (Zahlen, usw.) beschrieben werden können, sondern sie können auch dazu benutzt werden, ganze Objekthierarchien aufzubauen. Zum Beispiel können die Bedingungen, unter denen Komponenten mit Daten versorgt werden, mittels Konfigurationsdatei festgelegt werden. Eine Änderung der Bedingungen (z.B. zeitliche Verzögerung eines Modellaufrufs) kann durch Neukonfiguration des Softwaresystems ohne eine Neuübersetzung berücksichtigt werden.

Um dies zu ermöglichen, stellt das Framework einen speziellen Mechanismus zur Verfügung, der die Erzeugung von Objekten bloß aus dem Klassennamen (Typ) erlaubt. Ein solcher Mechanismus wird von der Programmiersprache C++, in der das REFORM-Framework erstellt wurde, nicht zur Verfügung gestellt. Er wurde im Framework durch Rückgriff auf Metainformation nachgebildet. Der Typ von Parameterobjekten muß also nicht bereits zur Übersetzungszeit feststehen, sondern wird in einer Konfigurationsdatei spezifiziert. Diese Datei wird grammatikgesteuert gelesen, und dabei werden die Objekte entsprechenden Typs erzeugt.

Man definiert in der Konfigurationsdatei alle Daten, mit denen die Komponente versorgt werden soll und gleichzeitig, wann die Daten aufgezeichnet und wann sie an die Komponente weitergeleitet werden sollen. Das Framework erzeugt aufgrund dieser Informationen die entsprechenden Objekte (für die Datenaufnahme und für den ereignisgesteuerten Ablauf), die letztendlich dafür verantwortlich sind, daß die Komponente mit der gewünschten Information versorgt wird.

Abbildung 2 zeigt die Zuordnung von Konfigurationsdateien (in der Abbildung mit „param“ bzw. „templ“ bezeichnet) zu den Komponenten. Komponenten können sich Konfigurationsdateien auch teilen (siehe Abb. 2 MMI und Kern), oder auf mehrere Konfigurationsdateien Bezug nehmen (siehe z.B. MMI in Abb. 2).

Auch die von der Kernkomponente der Automatisierung benötigte Information über die Anlagenkonfiguration (= Anordnung und hierarchische Gliederung der Aggregate) wird in einer Konfigurationsdatei gehalten. Daher ist die Anpassung der Architektur gemäß den Anforderungen einer speziellen Anlage zum Großteil ohne Änderungen des Quelltextes durchführbar. Anforderungen bzw. Änderungswünsche, die so massiv vom typischen Verhalten abweichen, daß die Konfigurationsmöglichkeiten nicht mehr ausreichen, können im objektorientierten Sinne durch Ableiten neuer Klassen implementiert werden. Die Schnittstellen der Komponenten bleiben dabei unverändert.

Zusammenfassung

Ein großer Teil der Automatisierungssoftware, die heute eingesetzt wird, ist noch mit veralteten Methoden entwickelt worden. Die modulare Programmierung führt zwar zu besserer Strukturiertheit der Software und damit zur Verbesserung der Qualität, bringt aber wenig in bezug auf die Produktivität und Flexibilität. Die Anwendung objektorientierter Techniken alleine erlauben zwar eine Steigerung der „inneren“ Flexibilität, aber die vorhandenen Produktivitäts- und Qualitätssteigerungspotentiale werden damit nur unvollständig ausgeschöpft. Durch den gleichzeitigen Einsatz von objektorientierten Techniken und konfigurierbaren Komponenten werden die vorhandenen Potentiale wesentlich besser ausgeschöpft. Außerdem können dadurch verteilte Systeme auf sehr flexible Weise individuell gestaltet werden. Besonders der Konfigurierbarkeit von Komponenten kommt eine zentrale Bedeutung zu, denn ständige Änderungen an Komponenten oder am Framework wirken sich besonders nachteilig aus. Besonders hervorzuheben ist

außerdem, daß die Konfigurierbarkeit und die leichte Einfügbarkeit von Komponenten dazu genutzt werden kann, um Aufgaben von den Softwareexperten zu den Anlagenexperten zu verschieben. Das wirkt positiv sowohl auf die Produktivität als auch auf die Qualität der Softwareentwicklung, weil der oft schwierige Kommunikationsprozeß zwischen diesen beiden Gruppen wegfällt.

Die Implementierungsarbeiten für die wiederverwendbaren Komponenten sind abgeschlossen, die Implementierungsarbeiten für eine spezielle Anlagenkonfiguration sind im Gange. Die Evaluation der tatsächlichen Produktivitäts- und Qualitätssteigerungseffekte ist daher noch nicht abgeschlossen. Der derzeitige Stand der Arbeiten zeigt aber bereits, daß die Entwicklungszeit und die Entwicklungskosten bei der Verwendung der hier vorgestellten Architekturbausteine deutlich geringer ausfallen, als bei konventioneller Vorgehensweise.

Schrifttum

- [1] REFORM: A Reusable Framework for Rolling Mills, online at <http://www.ssw.uni-linz.ac.at/REFORM/home.html>, 1998.
- [2] Pirklbauer, K., Plösch, R., Weinreich, R.: Object-Oriented and Conventional Process Automation Systems“, Proceedings of 39th International Scientific Colloquium at TU Ilmenau, Bd. 3, (1994), S. 566 bis 571,
- [3] Siegel, J. CORBA - Fundamentals and Programming, New York, John Wiley & Sons, Inc., 1996.
- [4] Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design Patterns – Elements of Reusable Object-Oriented Software, Reading, Massachusetts, Addison-Wesley, 1995.
- [5] Pree, W.: Design Patterns for Object-Oriented Software Development, New York, ACM Press 1994.
- [6] Pomberger, G., Blaschek G.: Prototyping und objektorientierte Software-Entwicklung, 2., überarbeitete Auflage, Wien, Hanser 1996.
- [7] Pree, W.: What characterizes a (software) component? Software – Concepts & Tools, Vol. 19, 1, (1998), S. 49 bis 56.
- [8] Stritzinger, A.: Komponentenbasierte Softwareentwicklung, Bonn, Addison Wesley 1997.
- [9] Sametinger, J.: Software Engineering with Reusable Components, Springer Verlag, 1997.