Java am Steuer

Eine Hybrid-Maschinensteuerung mit J4A

Ralf Röhrig, Beat Neuenschwander, Andres Koch

J4A ist eine Softwarearchitektur, welche für die Steuerung von Produktionsmaschinen zur Herstellung von großen Druckzylindern für die Tiefdruckindustrie entwickelt wurde. Darin werden Java (JSE) als High-Level-Komponente und ein Soft-SPS-System für den mit Personenschutz abgesicherten Realtime-Teil der Maschine eingesetzt. Die Softwarearchitektur ist skalierbar und wird für die verschiedenen unterschiedlichen Maschinen, vom Galvanisierungsbad, der Chemieversorgung, über die Schleif- und Poliermaschine bis hin zur High-Tech-Laser-Graviermaschine, eingesetzt.

Überblick und spezielle Randbedingungen

Vor einigen Jahren wurde die Softwareentwicklung der einzelnen Produktlinien bei der MDC Max Daetwyler AG in Bleienbach zusammengelegt. Der Grund dafür war, die vorhandene Kompetenz und das Know-how der einzelnen Entwickler für die Realisierung einer neuen Steuerungsgeneration zu nutzen, welche einheitlich für den gesamten Maschinenpark der Firma zum Einsatz kommen sollte. Eine der zentralen Aufgaben des neuen Konzeptes war, die vorhandenen Synergien der einzelnen Maschinen besser zu berücksichtigen und so den Entwicklungsaufwand und damit die „Time to Market" deutlich zu reduzieren.

Neben der Personensicherheit und Echtzeitfähigkeit für das direkte Maschinenumfeld waren auch Eigenschaften wie diverse Kommunikationsmöglichkeiten (z. B. Schnittstellen zu Leitsystemen und anderen Maschinen), variable Prozessgestaltung für die Ablaufsteuerung, eine einfache und umfassende Parametrierbarkeit (z. B. Meldungen, Maschinenparameter, Mehrsprachigkeit usw.) sowie ein einheitliches Erscheinungsbild nach außen (Human Machine Interface, kurz HMI) gefordert. Eine weitere Anforderung war die Möglichkeit, die einzelnen Ebenen auf unterschiedliche Rechner zu verteilen (z. B. wegen besserer Performance).

Mit der Entwicklung eines neuen Frameworks für die gesamte Maschinenpalette sollten auch einheitliche Entwicklungsmethoden und -plattformen definiert und eingeführt werden. Die Vorgabe war, auf dem Markt gängige Tools, Methoden und Programmiersprachen einzusetzen und die Neuentwicklung somit auf einen modernen Standard zu bringen.

Weitere Anforderungen an die neue Steuerungsgeneration:

- Schulungsaufwand beim Techniker und Kunden reduzieren,
- Aufwand für die Wartung der Software minimieren,
- Lizenzkosten für Software verringern,
- Verteilbarkeit auf einem oder mehreren Rechnern, je nach Leistungsbedarf.

Softwarearchitektur

Die genannten Randbedingungen führten bald zu einer Architektur in drei Schichten (s. Abb. 1):

- Fassade (Facade), als genereller Schnittstellen-Layer zum HMI, Leitsystem und anderen externen Maschinen.
- High-Level-Steuerung (HLS), welche die Kontrollfunktionen, die eigentliche Applikationslogik bis hin zur Ablaufsteuerung von Maschinenfunktionen beinhaltet.
- Low-Level-Steuerung (LLS), die die maschinennahen Funktionen, welche Personensicherheit und Echtzeitanforderungen erfüllen, enthält.

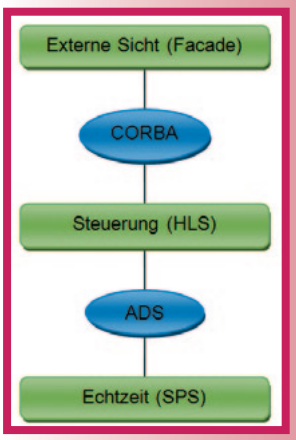

Abb. 1: J4A-Architektur

Die Facade und die HLS wurden für Java konzipiert, während die LLS durch eine Soft-SPS (in Software emulierte speicherprogrammierbare Steuerung) als Komponente standardmäßig von einem Hersteller (zurzeit Beckhoff Automation GmbH) eingekauft wurde.

Das Zusammenspiel zwischen Facade und HLS ist nach dem Observer-Muster entworfen, sodass der HLS-Teil keine Kenntnis vom angeschlossenen Subscriber haben muss. Die im HLS-Teil eingebaute Flowcontrol-Engine erlaubt, ein flexibles und unter Umständen kundenspezifisches Verhalten kontrolliert zu realisieren.

Dank der netzwerkfähigen Schnittstellen zwischen Facade und HLS (CORBA), sowie HLS und LLS (Sockets), kann, je nach Leistungsanforderung und Verteilungsvorgabe, jede Schicht auf einem eigenen Rechner ablaufen. Diese Trennung kann besonders zwischen HLS und LLS sinnvoll sein. Komplexere Maschinen, wie z. B. zur Laser-Gravur, erfordern mehr Berechnungsleistung als z. B. ein Galvanikbad. Mit Festlegung der Architektur wurde entsprechend auch die Projektorganisation aufgebaut und die Spezialisten eingesetzt. Zudem entwickelte sich eine klare Terminologie und Funktionsabgrenzung, wo was gelöst werden soll.

Schnittstellen nach außen (Facade)

Die oberste Ebene (Facade) ist für die externe Sicht auf die Maschine bestimmt. Diese enthält neben dem HMI verschiedene Adapter zur Kommunikation mit anderen Maschinen und/oder dem Leitsystem. Zwischen der Facade und der Steuerungsebene (HLS) kommt eine CORBA-Schnittstelle zum Einsatz. Die HLS dient hier als Server und die Elemente der Facade verbinden sich als Clients. Jeder Client kann sich zusätzlich als Listener am Server registrieren und somit auf Events reagieren, welche die HLS generiert. Wie viele Clients eines bestimmten Typs (z. B. HMI) sich auf den Server verbinden können und welche Zugriffsrechte diese besitzen, können serverseitig konfiguriert werden.

In der HLS registrieren sich die einzelnen Bausteine (z. B. Parameter, ProcessHandling, UIAccess, s. Abb. 2) der Steuerungsebene als sogenannte Komponenten. Diese Komponenten besitzen, aus der Sicht des Clients, Befehle (Operations) und Eigenschaften (Attributes). Der Client kann nun die definierten Befehle absetzen oder die Eigenschaften der Komponenten abfragen. Die erlaubten Befehle werden pro Komponente in einer XML-Datei definiert und müssen dann in dieser entsprechend ausprogrammiert werden.

Weitere Methoden, die von der Schnittstelle zur Verfügung gestellt werden, sind:

- Abfrage aller Komponenten, die an der HLS registriert sind,
- Abfrage aller Attribute-Namen, die eine Komponente zur Verfügung stellt,
- Abfrage bestimmter Attribute einer Komponente,
- Abfrage von Meta-Daten (definierte Operations) einer Komponente,
- Löschen und Editieren von Attributen einer Komponente.

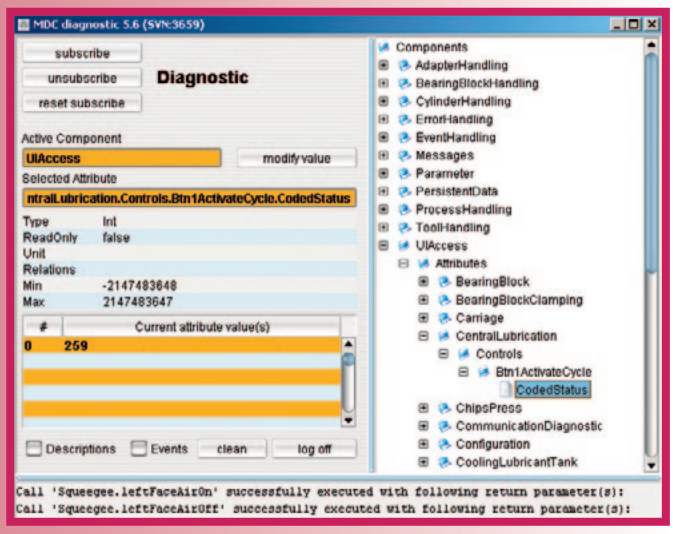

Abb. 2: Diagnose-Tool

Die so aufgebaute Schnittstelle ist äußerst flexibel und realisiert eine gute und einheitliche Anbindung der verschiedenen Facade-Clients. Sie ermöglicht außerdem den dynamischen Aufbau eines Diagnose-HMI (s. Abb. 2), mit dem die Funktionalität der Maschine unabhängig von den dort registrierten Komponenten möglich ist. Eine Maschine kann damit bereits in Betrieb genommen werden, bevor das eigentliche HMI gestaltet wurde.

Für das HMI wurde ein eigenes Swing-Framework entwickelt, welches die gemeinsame Grundfunktionalität (z. B. Editoren, Betriebsarten, Meldungs- und Hilfefenster, Einstellungs- und Kalibrierungsfenster) der J4A-Oberfläche zur Verfügung stellt. Die auf dem HMI angezeigten Kontrollelemente werden zum Großteil von der HLS gesteuert. Diese Steuerung beschränkt sich dabei auf Eigenschaften wie die Sichtbarkeit, die Bedienbarkeit, das Schaltverhalten (tastend/rastend) und den Status (aktiv/inaktiv).

Die Eigenschaften (Attributes) der Kontrollelemente sind in einer eigenen HLS-Komponente (UIAccess) als codierter Status (ein Integer-Wert, s. Abb. 3) hinterlegt und werden dort zyklisch abgefragt. Dort sind ebenfalls die zulässigen Aktionen auf den einzelnen Devices definiert, die von der HLS ausgeführt werden dürfen.

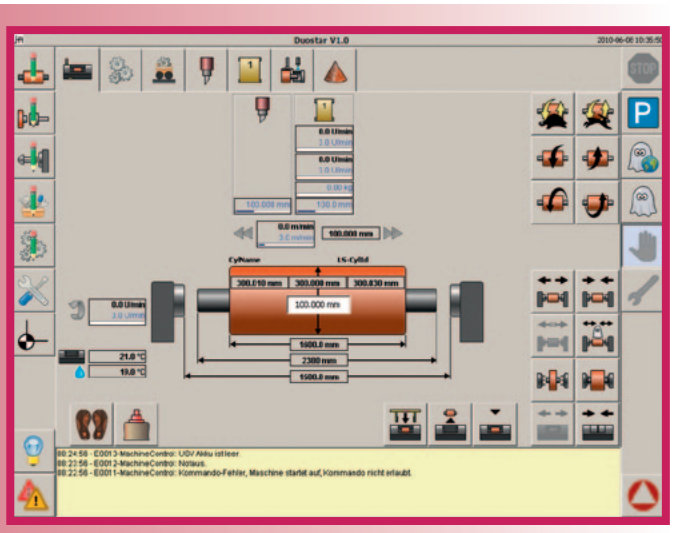

Abb. 3: J4A-HMI (für die Produktionsmaschine Duostar)

HMI-seitig werden J4A-spezifische Kontrollelemente als Ableitung von den Swing-Controls zur Verfügung gestellt. Ein J4A-Button benötigt somit nur noch wenige Propertys (DeviceName, AttributName, PressOperation, ReleaseOperation).

High-Level-Steuerung (HLS)

Die mittlere Ebene steuert zum einen den Ablauf der Maschinenprozesse und stellt zum anderen die Schnittstelle zur SPS und der Facade zur Verfügung. Mit ihr wird die Maschine konfiguriert und sie hält alle benötigten Parameter und Meldungen bereit. Sie verteilt außerdem auftretende Events an alle registrierten Komponenten und organisiert die Fehlerbehandlung. Devices und Aktivitäten, die in mehreren Maschinen verwendet werden können, sind in einer allgemeinen Maschinenbibliothek hinterlegt.

Jedes J4A-Projekt beinhaltet oben beschriebene Standardkomponenten sowie einen maschinenspezifischen Teil (s. Abb. 4). Die Standardkomponenten sind jeweils in eigenen Teilprojekten organisiert, welche dann als Bibliotheken in das Maschinenprojekt integriert werden können.

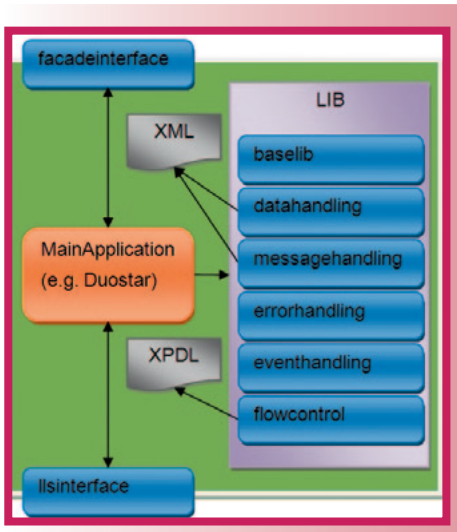

Abb. 4: HLS-Struktur (Komponenten)

In der HLS unterscheiden wir zwischen Prozessen und Aktivitäten. Eine Aktivität ist Teil eines Bearbeitungsprozesses und stellt eine wiederverwendbare logische Einheit innerhalb eines Prozesses dar. Jede Maschine besitzt einen definierten Vorrat an Aktivitäten, die zu einem Prozess zusammengesetzt und von dieser ausgeführt werden können. Jede Aktivität gleicht einer State-Machine, die in Abhängigkeit ihres Zustandes die entsprechenden Befehle der Devices aufruft. Die Prozessablaufsteuerung wiederum steuert den Aufruf der für einen bestimmten Prozess definierten Aktivitäten in der entsprechenden Reihenfolge. Für jede Aktivität können Parameter definiert werden. Außerdem kann definiert werden, welche parallelen und seriellen Aktivitäten vor und nach einer Aktivität zugelassen oder verboten sind.

Schnittstelle zur SPS-Steuerung

Zwischen der Steuerungsebene (HLS) und der Echtzeitebene (SPS) kommt ebenfalls eine TCP/IP-Schnittstelle zum Einsatz. Die HLS verbindet sich in diesem Fall als Client mit dem ADS-Server (Automation Device Specification), der ein Bestandteil der Soft-SPS ist.

In der Echtzeitebene werden die einzelnen Baugruppen (z. B. Tür, Schleifeinheit) einer Maschine implementiert. Eine Baugruppe (Device) stellt Befehle (z. B. Tür auf/zu) und Meldungen (z. B. Time-out beim Öffnen) zur Verfügung. Aus einer XML-Datei, welche die SPS zur Verfügung stellt, werden die Device-Proxy-

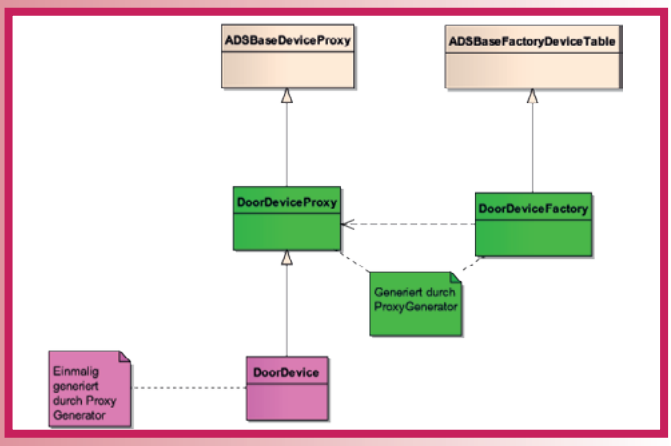

Abb. 5: Framework für generierte Device-Proxys

Klassen für die HLS generiert. Diese ermöglichen der HLS via ADS-Server auf die Konfiguration und den Status der Devices in der SPS zuzugreifen.

Eine vom Device-Proxy abgeleitete Device-Klasse stellt alle Methoden und Eigenschaften zur Verfügung, die innerhalb der HLS für dieses Device aufrufbar sein sollen (s. Abb. 5). Die verfügbaren Kommandos und deren Parameter sind wiederum in einer XML-Datei definiert.

Prozessablaufsteuerung

Das Herzstück der Prozessablaufsteuerung ist eine Workflow-Engine, die es ermöglicht, nach dem XPDL (XML Process Definition Language)-Standard definierte Prozesse abzuarbeiten. XPDL ist eine von der WfMC (Workflow Management Coalition) definierte Sprache zur Beschreibung von Geschäftsprozessen. Die Prozesse werden mit dem Open-Source-Workflow-Editor JaWE erstellt und als XML-Datei abgespeichert. Sie können die Elemente Start, End, Split, Join, Activity und Sub-Process enthalten. Jeder Prozess kann in einem anderen Prozess als Sub-Process eingefügt werden, wodurch ein modularer Aufbau möglich ist.

Die Schnittstelle zum Code bildet das Activity-Element. Anhand der ID wird in der HLS mittels einer Factory (Factory-Muster) die entsprechende Aktivitäts-Klasse instanziiert. Jede Aktivitäts-Klasse muss ein bestimmtes Interface implementieren, dessen wichtigsten Methoden `doPreCondition()`, `doProcessing()` und `doPostCondition()` sind. Diese werden während der Ausführung der Workflow-Engine aufgerufen.

Mit diesem Framework können Prozesse grafisch zusammengebaut werden. Der große Vorteil davon ist, dass Änderungen am Ablauf nicht zwingend Code-Änderungen verursachen.

Die Erfahrung hat gezeigt, dass die Funktionalität nicht auf zu viele Aktivitäten aufgeteilt werden darf, da ansonsten die Interaktion der Aktivitäten über Events oder den gemeinsamen Variablen-Pool den Entwicklungsaufwand und die Komplexität deutlich erhöht.

Eingesetzte Technologien

Im Folgenden soll nochmals ein Überblick über die verwendeten Technologien und Tools gegeben werden:

- EAP 8.0 von SparxSystems ist das UML-Designer-Tool unserer Wahl.
- Bei der Softwareentwicklung der Facade und der HLS setzen wir auf die JDK 1.6 (Java SE 6) in Verbindung mit Eclipse 3.5 und Netbeans 6.7 für die Erstellung des HMI mit Swing.
- Die Soft-SPS-IDE TwinCAT kommt von der Firma Beckhoff.
- Alle Ebenen und Teilprojekte unterliegen der Versionsverwaltung (SVN).

Folgende Third-Party-Java-Bibliotheken werden bei uns unter anderem eingesetzt:

- JacORB für das CORBA-Interface und
- Log4j für das Logging.

Lesson learned

Das J4A-Framework hat sich in den letzten Jahren, in denen diverse Maschinen und Anlagen damit realisiert wurden, sehr gut etabliert. Trotz einer langen Durststrecke während der aufwendigen Analyse- und Designphase und der Implementierung des Frameworks wurde das Projekt ein voller Erfolg. Mittlerweile konnten sogar externe Produkte mit der Plattform realisiert werden.

Die Stärken der Anwendung liegen sicher in der Flexibilität und der Wiederverwendbarkeit von Baugruppen (Devices). Gerade bei einer Maschinenpalette, in der viele Gemeinsamkeiten der einzelnen Maschinen untereinander vorhanden sind, kann das Framework seine Vorteile perfekt zur Geltung bringen.

Das Projekt hat uns natürlich auch persönlich einen enormen Gewinn gebracht. Alle Entwickler, die von Anfang an dabei waren, kamen aus der prozeduralen Softwareentwicklung. Nur durch die Einführung von OO-Methoden und mithilfe intensiver Unterstützung des externen Technologie-Coaches konnte das Projekt so realisiert werden, wie es sich momentan präsentiert.

Ralf Röhrig, Dipl.-Inf. (FH), Daetwyler Industries, entwickelt und betreut in dem Projekt J4A die Facade-Komponenten sowie Teile der High-Level-Ebene. Er ist von der ersten Stunde an in dieses Projekt involviert. Neben seiner Entwicklertätigkeit leitet er die Ausbildung in der Abteilung Forschung und Entwicklung.
E-Mail: rroehrig@daetwyler.com

Beat Neuenschwander, Eidg. dipl. Softwareentwickler MAS-IT (Master of Advanced Studies), Daetwyler Industries, entwickelt und betreut in diesem Projekt die High-Level-Komponenten, hauptsächlich die Schnittstelle zur SPS und die Prozesssteuerung der Maschine. Eine weitere Tätigkeit ist das Erstellen von Anforderungsspezifikationen in Zusammenarbeit mit den Kunden.
E-Mail: bneuenschwander@daetwyler.com

Andres Koch, dipl. El. Ing. HTL, M. Math, Object Engineering GmbH, hatte in diesem Projekt die Rolle als Technologie-Coach und Softwarearchitekt und brachte seine über 30-jährige Erfahrung in Softwaretechnik im industriellen und kommerziellen Umfeld mit ein. Er arbeitet als Softwarearchitekt, spezialisiert auf verteilte Systeme, der Modernisierung von Software und auf Programmiersprachen (DSLs), bei der Object Engineering GmbH. Er ist als Lehrbeauftragter am MAS-Studium für Software-Engineering der Hochschule für Technik in Rapperswil tätig.
E-Mail: akoch@objeng.ch