End User Development für Leitstand-Arbeitsplätze: Beschreibung einer Systemarchitektur

Stefan Hofer, Sven Wende, Alexander Will, Heinz Züllighoven

C1 WPS GmbH
Vogt-Kölln-Straße 30
D-22527 Hamburg
[stefan.hofer|sven.wende|alexander.will|heinz.zuellighoven]@c1-wps.de

Abstract: End-User Development beschäftigt sich mit der Frage, wie Endbenutzer die von ihnen eingesetzte Software anpassen können. Dazu muss die Software eine geeignete Architektur aufweisen. Dieser Beitrag skizziert eine solche Architektur für einen Leitstand-Arbeitsplatz zur Überwachung und Steuerung technischer Anlagen.

1 Einleitung

Die Veränderung oder Anpassung von Software durch ihre Benutzer, das sog. *End-User Development*, gewinnt im Rahmen der Anwendungsorientierung zunehmend an Bedeutung. Wenn Software durch Endbenutzer anpassbar sein soll, dann muss diese Eigenschaft bereits in der Entwicklung berücksichtigt werden. Anpassbarkeit wirkt sich nicht lediglich auf der Ebene der Interaktionskonzepte aus, sondern bestimmt substanziell die grundlegende Softwarearchitektur.

In diesem Beitrag skizzieren wir zunächst den Diskussionsstand um End-User Development. Dann beschreiben wir eine Architektur für einen Leitstand-Arbeitsplatz zur Überwachung und Steuerung technischer Anlagen, die die Anforderung nach Anpassbarkeit durch den Endbenutzer frühzeitig berücksichtigt. Abschließend zeigen wir, wie sich die Anforderungen an die Architektur einer anpassbaren Software mit Hilfe des Eclipse Komponentenmodells und der Eclipse Rich Client Platform erfüllen lassen.

Das Beispiel des Leitstand-Arbeitsplatzes stammt aus einem professionellen Projektkontext innerhalb des BMBF-Projekts „Component-based End User Development (CoEUD)“ [Co06]. Um die kommerzielle Einsetzbarkeit der erarbeiteten EUD-Konzepte nachzuweisen, entwickelt die C1 WPS GmbH gemeinsam mit dem Deutschen Elektronen Synchrotron (DESY) und der Universität Hamburg ein anpassbares Kontrollsystem für die Anlagen des DESY [CSS06].

2 End User Development

Unter dem Begriff *End User Development* (*EUD*) werden Methoden, Techniken und Werkzeuge zusammengefasst, mit denen Endbenutzer Software verändern und erweitern können [LPKW06]. Unter einem *Endbenutzer* (*End User*) wird dabei eine Person verstanden, die direkt mit einem Softwaresystem interagiert [Kl03]. Eine scharfe Trennung der englischen Begrifflichkeiten *End User* und *User* wird im Allgemeinen nicht vorgenommen – im Gegenteil werden diese Begriffe häufig synonym verwendet.

Einen detaillierten Überblick über den aktuellen Stand auf diesem Forschungsfeld liefern u. a. [Eu03], [LPRW03] und [LPW06].

EUD liefert einen Lösungsansatz für verbreitete Probleme der Anwendungsentwicklung. Dazu gehören die Anpassung von Software an Individuen und Benutzergruppen sowie der Umgang mit unvorhergesehenen Anforderungen.

- Selbst in klar definierten Domänen unterscheiden sich Endbenutzer hinsichtlich Fähigkeiten, Wissen und Kultur. Daraus ergeben sich individuell verschiedene Arten, mit Software zu interagieren [CPFM03]. Lässt sich Software an persönliche Bedürfnisse und Vorlieben anpassen, steigt ihre Akzeptanz und die Benutzung wird erleichtert.

- Trotz ihrer individuellen Unterschiede werden Endbenutzer oft in Benutzergruppen mit typischen Aufgaben und Fähigkeiten eingeteilt. Die gruppenspezifischen Bedürfnisse werden häufig entweder durch Parametrisierung der Applikation oder durch Bereitstellung diversifizierter Produkte (etwa in Form von Produktlinien) befriedigt. EUD eröffnet eine weitere Möglichkeit, eine Anwendung flexibel an Benutzergruppen anzupassen.

- Im Lebenszyklus einer Software ergeben sich häufig fachliche Änderungen und Erweiterungen der Anforderungen. EUD kann Endbenutzer dabei unterstützen, einen Teil dieser Anforderungen kostengünstig selbst umzusetzen [Kl03].

EUD setzt veränderbare und erweiterbare Software voraus. Diese Eigenschaft von Software ist unter englischen Begriffen wie *adaptable*, *customizable*, *tailorable* oder deren deutscher Entsprechung *anpassbar* bekannt [St02]. Die dabei verwendeten Techniken werden nach [Mø95] in drei Stufen eingeteilt, die wir im Weiteren verwenden werden:

- Bei der Konfiguration (*customization*) wird Software durch Auswählen aus einer vordefinierten Menge von Optionen angepasst.

- Bei der Integration (*integration*) wird der Software vorgefertigte Funktionalität hinzugefügt.

- Bei der Erweiterung (*extension*) wird neue Funktionalität für die Software entwickelt, beispielsweise durch das Schreiben von neuem Quelltext.

3 Anforderungen an einen anpassbaren Leitstand-Arbeitsplatz für technische Anlagen

In diesem Abschnitt beschreiben wir die Anforderungen an einen anpassbaren Leitstand-Arbeitsplatz für die Überwachung und Steuerung technischer Anlagen. Wir charakterisieren zunächst den Nutzungskontext einer solchen Anwendung und beziehen uns dabei auf die konkreten Gegebenheiten am Deutschen Elektronen Synchrotron (DESY). Das DESY ist eine Forschungseinrichtung mit Standorten in Hamburg und Zeuthen, in der Beschleunigeranlagen für die Forschungsschwerpunkte Teilchenphysik und Photonenforschung eingesetzt werden. Beschleuniger sind hochkomplexe, aus vielen einzelnen Geräten zusammengesetzte, technische Anlagen, mit denen Partikel durch starke Elektromagneten beschleunigt werden, um sie dann kollidieren zu lassen. Um den reibungslosen Betrieb dieser Anlagen sicherzustellen, müssen die einzelnen Geräte ständig überwacht und gesteuert werden.

3.1 Charakterisierung des Nutzungskontextes

Für die Überwachung der technischen Anlagen am DESY sind Operateure verantwortlich. Informationen über den Zustand der Anlagen werden über ein Kontrollsystem bezogen, das auch direkt auf einzelne Geräte zugreifen kann. So können Informationen (z. B. aktuelle Messwerte) abgefragt und Steuersignale abgesendet werden. Das Kontrollsystem stellt die technische Infrastruktur zur Unterstützung der Operateure bereit. Ein einzelner Messpunkt eines bestimmten Gerätes wird als Prozessvariable bezeichnet (beispielsweise der Druck in einer bestimmten Pumpe). In Einrichtungen von der Größe des DESY existieren mehrere hunderttausend Prozessvariablen in verschiedenen Bereichen der Anlage. Die einzelnen Operateure sind meist für einen abgegrenzten Bereich (z. B. Kältetechnik oder Stromversorgung) zuständig. Daher ist für einen Operateur normalerweise nur eine Teilmenge der verfügbaren Prozessvariablen interessant.

Der Arbeitsplatz eines Operateurs sollte Maschinen und Anlagen beliebiger Hersteller überwachen und steuern können. Das Benutzungsmodell [Zü05] orientiert sich dabei an der Metapher eines Leitstandes. Laut DIN 19222 dient ein Leitstand (bzw. eine Leiteinrichtung) dazu, die Gesamtheit aller Maßnahmen zu leiten, „die einen im Sinne festgelegter Ziele erwünschten Ablauf eines Prozesses bewirken“ [DIN19222].

Diese Metapher soll ausdrücken, dass eine entsprechende Anwendungssoftware von den Endbenutzern als ein Instrumentarium verwendet wird, um auf Informationen der zu überwachenden Geräte zuzugreifen und diese zu steuern. Dabei steht der Gedanke im Vordergrund, dass eine solche Software eine Infrastruktur vorgibt, in die sich spezialisierte Anwendungen für die Steuerung und Überwachung von Anlagen integrieren können – ebenso wie reale Leitstände verschiedene Instrumente in einem gemeinsamen Rahmen zusammenfassen. Neben der Anlagensteuerung erlauben reale Leitstände häufig auch die Simulation verschiedener Betriebsszenarien, um z. B. das Betriebspersonal zu schulen. In einem realen Leitstand sind verschiedene Elemente integriert, die dem Endbenutzer spezielle Funktionen anbieten. Ein wichtiges Merkmal ist dabei, dass die einzelnen Funktionselemente in einer festgelegten Topologie angeordnet sind.

Diese Merkmale sollten insgesamt auch für einen Software-Leitstand, im weiteren Leitstand-Arbeitsplatz genannt, gelten. Die Darstellung in Abbildung 1 deutet an, dass die Benutzungsoberfläche eines Leitstand-Arbeitsplatzes aus den Repräsentationen einzelner Funktionselemente zusammengesetzt ist.

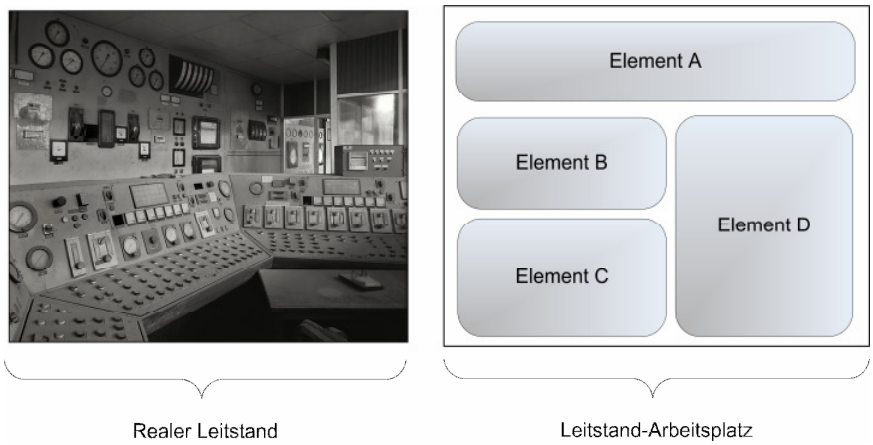

Abbildung 1: Die Leitstand-Metapher

Elementare Funktionen eines Leitstandes sind beispielsweise die Anzeige von Messwerten in Instrumenten und Alarmierung bei kritischen Systemzuständen. Ein Leitstand enthält spezifische Baugruppen für diese Funktionsbereiche, denen im Leitstand-Arbeitsplatz Software-Elemente entsprechen. Der Anzeige von Messwerten in Instrumenten entspricht in einem Leitstand-Arbeitsplatz das Konzept der sog. synoptischen Displays. Unter einem synoptischen Display wird ein Arrangement von Grafikelementen verstanden, die jeweils die notwendigen Instrumente repräsentieren, um eine Anlage zu überwachen. Diese Instrumente zeigen die Daten bestimmter Prozessvariablen an und repräsentieren somit den aktuellen Zustand der Anlage.

Reale Leitstände dienen also dazu, komplexe technische Systeme zu überwachen und zu steuern. Ein softwarebasierter Leitstand-Arbeitsplatz, der einen Endbenutzer bei dieser Arbeit unterstützt, muss daher vor allem intuitiv bedienbar und in einem hohen Maße auf die Arbeitsaufgaben zugeschnitten sein. Behinderungen des Arbeitsflusses durch nicht-intuitive Software führen nicht zwangsläufig zu Fehlern, aber auf jeden Fall zur Frustrationen des Benutzers. Im vorliegenden Nutzungskontext ist besonders hervorzuheben, dass eine Anlage nur in dem Umfang überwacht werden kann, der durch den Leitstand-Arbeitsplatz abgedeckt ist. Werden Geräte nicht durch die Software repräsentiert, entziehen sie sich der Überwachung.

3.2 Anpassungsbedürfnisse der Endbenutzer

Grundsätzlich soll ein Leitstand-Arbeitsplatz individuell anpassbar sein. In bestimmten Situationen können aber auch organisatorische Gründe gegen eine individuelle Anpassung sprechen, insbesondere wenn derselbe Arbeitsplatz von mehreren Personen genutzt wird. Während klassische Leitstände i. d. R. fest zusammenhänge Konstruktionseinheiten bilden, bietet ein softwarebasierter Leiststand-Arbeitsplatz die Möglichkeit, individuelle Anpassungsbedürfnisse und organisatorische Rahmenbedingungen in Einklang zu bringen. Dies wird am Beispiel der Überwachungsbereiche der Endbenutzer verdeutlicht:

- *Individuelle Überwachungsbereiche:* Ein Endbenutzer überwacht die Teile einer Anlage, die direkt von seinem persönlichen Tätigkeitsfeld abhängen. Dies ist so individuell, dass es keinen weiteren Endbenutzer mit identischem Überwachungsbereich gibt. Der entsprechende Leitstand-Arbeitsplatz sollte ebenfalls individuell anpassbar sein.

- *Gruppeneinheitliche Überwachungsbereiche:* Abhängig vom Aufbau einer Anlage gibt es Gruppen von Endbenutzern, die sich einen (möglicherweise größeren) Überwachungsbereich teilen. Innerhalb dieser Gruppen verwenden alle Mitglieder eine identisch oder gleichartig angepasste Arbeitsumgebung. Beispiele sind große Funktionsbereiche wie Stromversorgung oder Kältetechnik. Der Umfang der individuellen Anpassungsmöglichkeiten wird hierbei durch das organisatorische Umfeld vorgegeben.

Diese generelle Forderung nach Anpassung kann konkretisiert werden. Es können feingranulare Anpassungsanforderungen abgeleitet werden, die den von [Mø95] beschriebenen Stufen entsprechen:

- *Konfiguration des Leitstandes im Rahmen einer vorgegebenen Leitstand-Funktion:* Anpassungen auf dieser Ebene bewegen sich innerhalb der vorgegebenen Einstellmöglichkeiten einer bestimmten Leitstand-Funktion. Die Realisierung kann beispielsweise mit Mitteln der direkten Manipulation bzw. über Dialoge und Formulare erfolgen. Programmierkenntnisse sollten seitens der Endbenutzer hierfür nicht erforderlich sein.

- *Integration von vorgegebenen Funktionen in den Leitstand:* Innerhalb des persönlichen Leitstand-Arbeitsplatzes werden einem Endbenutzer verschiedene Funktionen angeboten. Die Anpassbarkeit besteht auf dieser Ebene darin, dass weitere Funktionen hinzugefügt oder bestehende entfernt werden können – mit den damit verbunden Auswirkungen auf die Benutzungsoberfläche des Gesamtsystems. Dies kann über Konfigurationsmechanismen innerhalb der Leitstand-Software realisiert werden – hierfür sollten ebenfalls keine Programmierkenntnisse erforderlich sein.

- *Erweiterung des Funktionsumfangs des Leitstandes:* Der persönliche Leitstand-Arbeitsplatz wird hierbei nicht nur um bereits existierende Funktionen ergänzt, sondern durch die Entwicklung gänzlich neuer Funktionen erweitert. Dieser Typ von Erweiterungen erfordert letztlich die programmiertechnische Implementierung neuer Funktionalitäten. Die Software-Architektur des Leitstand-Arbeitsplatzes muss hierfür geeignet gestaltet sein. Leitstand-Funktionen müssen so entwickelt werden, dass sie konsistent in das Gesamtsystem integriert werden können. Für Erweiterungen auf dieser Ebene sind Programmierkenntnisse erforderlich.

4 Die Architektur eines anpassbaren Leitstand-Arbeitsplatzes

Anpassbarkeit eines Systems bedeutet für uns mehr, als entsprechende Interaktionskonzepte bereit zu stellen. Entscheidend ist, dass die softwaretechnische Architektur die Anpassbarkeit frühzeitig berücksichtigt. Nur wenn dies schon bei der Gestaltung der Softwarearchitektur geschieht, kann eine solide und nachhaltig anpassbare Software entstehen. Im weiteren Verlauf dieses Beitrags beschreiben wir eine Architektur, die diesen Anforderungen für den Einsatzkontext eines Leitstand-Arbeitsplatzes gerecht wird. Dazu werden zunächst die Merkmale der Architektur einzeln erläutert und abschließend zu einer Architekturskizze zusammengeführt.

4.1 Komponenten-basierte Integrationsplattform

Zahlreiche Softwareprojekte haben gezeigt, dass sich Anforderungen des EUD wie Erweiterbarkeit und Änderungsfreundlichkeit mit Hilfe von Software-Komponenten verwirklichen lassen [St02, Mø04]. Wir folgen dem Komponentenbegriff nach [Sz98] und verstehen darunter eine Entwurfseinheit, die klar definierte Schnittstellen und Abhängigkeiten aufweist und separat ausgeliefert werden kann. So können Komponenten verschiedener Hersteller miteinander verbunden und eingesetzt werden.

In [St02] wird beschrieben, wie sich die beiden EUD-Stufen *Integration* und *Erweiterung* mit Komponenten umsetzen lassen:

- *Integration*: Anpassung wird durch unterschiedliche Komposition von Komponenten realisiert. Komponenten können in eine Komposition eingefügt, daraus entfernt oder anders miteinander verbunden werden.

- *Erweiterung*: Neu geschriebene Komponenten erlauben, unter Wahrung festgelegter Schnittstellen neue Funktionalität zu implementieren.

In einer beliebigen Zusammenstellung von Komponenten integrieren sich diese nicht zu einer konsistenten Anwendung. Dieses Problem kann eine *Integrationsplattform* (siehe Abbildung 2) lösen, die grundlegende Architekturentscheidungen vorgibt und so ein gewisses Maß an Qualität, Integrität und Nachhaltigkeit für das Gesamtsystem sicherstellt. In diesen Bereich der *technischen Integration* von Komponenten fällt auch die Verwendung einheitlicher Benutzungsoberflächen (siehe Abschnitt 4.2). Wird die Integrationsplattform für einen bestimmten Anwendungsbereich spezialisiert, lässt sich ein höherer Grad an Integration erreichen. Bei dieser *fachlichen Integration* wird die Plattform um fachlich motivierte Anwendungsmodelle (siehe Abschnitt 4.3) und Services (siehe Abschnitt 4.4) erweitert.

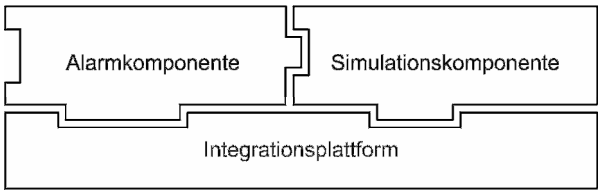

Abbildung 2: Integration von Komponenten

4.2 Benutzungsoberfläche

Die beschriebene Integrationsplattform lässt verschiedene Benutzungsoberflächen zu. Als Entwickler eines Leitstand-Arbeitsplatzes wünscht man sich aber ein Rahmenwerk, das die Implementierung einer Benutzungsoberfläche erleichtert. Im konkreten Anwendungsfall legt die Komplexität der erforderlichen Benutzungsoberfläche eine Rich-Client Lösung nahe. Eine Rich-Client Architektur erlaubt, die graphischen Möglichkeiten des Betriebssystems voll zu nutzen. Im Vergleich zu Thin-Clients können wesentlich komplexere Benutzungsoberflächen gestaltet werden, die besser an die Anforderungen der Endbenutzer angepasst werden können. Endbenutzer von Rich-Client Anwendungen sind zudem weitgehend unabhängig von der Qualität der (externen) Netzwerkverbindung und sie erhalten in der Regel unmittelbares Feedback über ihre Aktionen.

4.3 Anwendungsmodell

Eine wichtige Grundregel objektorientierter Anwendungssoftware ist deren Strukturierung anhand fachlicher Einheiten (*direct mapping rule* [Me97]). Dabei werden die für die Software relevanten Elemente des Anwendungsbereichs auf Elemente des Anwendungsmodells der Integrationsplattform abgebildet [Zü05]. Entwickler von Komponenten können somit Typen nutzen, die Fachbegriffe wie beispielsweise *Prozessvariable*, *Gerät*, *Alarm* oder *Benutzer* verkörpern.

4.4 Services

Die Verwendung von Services (*Diensten*) unterstützt die lose Kopplung und die Wiederverwendbarkeit einzelner Systemteile. Verschiedene Komponenten können die Funktionalität eines Services nutzen, ohne dessen technische Realisierung zu kennen. Wir unterscheiden zwischen technischen und fachlichen Services:

Technische Services

Technische Services kapseln die technische Infrastruktur einer Anwendung. Sie bieten grundlegende Dienstleistungen, wie z. B. Logging oder Zugriff auf Verzeichnisdienste an, die keinen unmittelbaren Bezug zu fachlichen Konzepten der Anwendung besitzen.

Fachliche Services

Unter fachlicher Funktionalität verstehen wir das in eine konkrete Implementation gefasste Anwendungswissen der Entwickler [Zü05]. Fachliche Services kapseln dieses Anwendungswissen. Die angebotene Dienstleistung kann z. B. in der Handhabung und Aufbewahrung von Gegenständen des Anwendungsmodells bestehen. Da die Komplexität des Service in dessen Implementation verborgen ist, wird die Dienstleistung für deren Nutzer leichter greifbar. Fachliche Services skalieren auf zwei Ebenen. Da sie den Teil einer Anwendung kapseln, der handhabungs- und präsentationsunabhängig ist, lassen sich zu einem Service prinzipiell verschiedene Benutzungsschnittstellen entwerfen. Zudem kann die Anwendung durch veränderte oder neue fachliche Services erweitert werden.

Entscheidend für die Güte der Architektur sind hierarchische Beziehungen zwischen den verschiedenen Service-Typen. Fachliche Services können sich gegenseitig verwenden. Um ihre Dienstleistung zu erbringen, können sie dabei auch auf technische Services zugreifen. Technische Services können sich ebenfalls gegenseitig benutzen, verwenden aber keine fachlichen Services oder fachlichen Kernkonzepte.

4.5 Architekturskizze eines Leitstand-Arbeitsplatzes

Die beschriebenen Merkmale sind in Abbildung 3 am Beispiel eines Leitstand-Arbeitsplatzes für das DESY zusammengefasst. Die Integrationsplattform stellt dabei nicht nur einen Kern zum Andocken von Komponenten dar, sondern bietet den Komponenten Zugriff auf zentrale Services, ein fachlich motiviertes Anwendungsmodell und einen Rahmen für die Integration ihrer Benutzungsoberfläche. Die Komponenten werden also technisch und fachlich integriert.

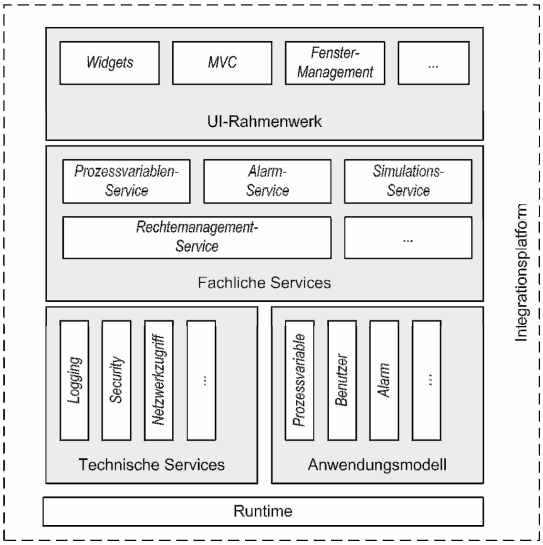

Abbildung 3: Die Architektur des Leitstand-Arbeitsplatzes

5 Die technische Umsetzung auf Basis von Eclipse RCP

Die Umsetzung der im vorigen Abschnitt beschriebenen Architektur erfordert eine Unterteilung des zu entwickelnden Systems in Komponenten. Zur softwaretechnischen Realisierung eignet sich die Eclipse Rich Client Platform (RCP). Dabei handelt es sich um ein frei verfügbares und vollständig in Java implementiertes Open-Source-Framework. Eclipse RCP ist aus der in Softwareentwicklerkreisen verbreitet eingesetzten Eclipse-IDE, einer Entwicklungsumgebung für Java-Anwendungen, entstanden und enthält insbesondere für die Gestaltung von Benutzungsoberflächen ein reichhaltiges Arsenal an Komponenten, auf die Entwickler bei der Gestaltung eigener Anwendungen zurückgreifen können.

Für die technische Ausgestaltung der Architektur bietet sich das Komponentenmodell von Eclipse [Ec06] an. Eclipse-basierte Anwendungen bestehen ausschließlich aus Plugins [GB04]. Diese kapseln eine bestimmte Funktionalität und lassen sich einzeln oder im Verbund (in Form so genannter *Features*) distribuieren. Die Abhängigkeiten (*Dependencies*) zwischen den einzelnen Plugins bilden dabei einen azyklischen gerichteten Graph und werden von einer minimalen *Runtime* verwaltet. Mit Hilfe dieses Komponentenmodells lässt sich, einen sinnvollen Schnitt der einzelnen Plugins oder Features vorausgesetzt, die Funktionalität einer Anwendung in Komponenten zerlegen, die einzeln an Endbenutzer weitergegeben werden können. Dies ermöglicht es Endbenutzern, bei Bedarf geeignete Komponenten auszuwählen und in ihr Anwendungssystem zu integrieren und dieses damit individuell an die eigenen Bedürfnisse anzupassen.

Die Eclipse-Architektur bietet darüber hinaus einen einfachen Erweiterungsmechanismus. Plugins können explizit Stellen definieren, an denen andere Plugins Funktionalität ergänzen können. Diese Stellen heißen Erweiterungspunkte (*Extension Points)*. Steuert ein anderes Plugin eine Implementation zu einem Erweiterungspunkt bei, so spricht man von einer Erweiterung (*Extension*). Entwickler von Komponenten können diesen Mechanismus nutzen, um fremde Anwendungen an vordefinierten Punkten zu erweitern und die eigene Anwendung explizit für Erweiterungen durch Dritte zu öffnen. Endanwender profitieren von dieser Möglichkeit i. d. R. durch die Integration von extern entwickelten Komponenten, die zusätzliche Funktionalität bieten. Es ist jedoch nicht ausgeschlossen, dass Endanwender entsprechende Erweiterungen auch aktiv selbst programmieren.

Seit Version 3.0 basiert die Plugin-Runtime von Eclipse auf dem Equinox OSGi-Framework [Eq06]. Die OSGi Alliance (Open Services Gateway Initiative), ein unabhängiges Non-Profit-Industriekonsortium, dem u. a. Sun Microsystems, IBM und Ericsson angehören, hat sich zum Ziel gesetzt, eine auf Java basierende Service- und Komponentenplattform zu spezifizieren, mit deren Hilfe die Interoperabilität von Anwendungen verbessert werden kann [OSGi06]. Das OSGi-Framework ergänzt Eclipse u. a. um eine Service-Infrastruktur mit flexiblen Mechanismen zum Auffinden von Serviceimplementierungen [HS05].

Darüber hinaus bietet es Funktionalitäten für das Management von OSGi-Bundles (Plugins), die u. a. auch zur Fernwartung eines Anwendungssystems genutzt werden können. Endbenutzer kommen mit OSGi nicht zwangsläufig direkt in Berührung – die Verwendung von OSGi-Features kann aber einen entscheidenden Beitrag zur internen Strukturierung der Anwendung leisten und damit die Grundlage für die individuelle Anpassbarkeit durch Endbenutzer legen.

Die große Stärke von Eclipse RCP liegt in der flexiblen Gestaltung graphischer Benutzungsoberflächen. Das Framework bietet ein großes Angebot an Oberflächen-Elementen, die sich für die Entwicklung von Anwendungen eignen. Die so genannte *generische Workbench* legt dabei das grundlegende Interaktionsmodell und die generelle Gestaltung der grafischen Benutzungsoberfläche aller RCP-basierten Anwendungen fest. Das Konzept der Workbench beruht auf dem Zusammenspiel von Editoren, Views und Perspektiven. Aus Sicht des Endbenutzers sind Perspektiven ein geeignetes Instrument zur individuellen Gestaltung der Benutzungsoberfläche. Benutzer können Views innerhalb einer Perspektive frei verschieben und relativ zu anderen Views anordnen. Darüber kann die Sichtbarkeit von Aktionen in Menüs und Werkzeugleisten einer Anwendung in Abhängigkeit von der aktuellen Perspektive gesteuert werden. Entwicklern von Komponenten bietet die Workbench die Möglichkeit, die Benutzungsoberfläche konsistenzwahrend mit Hilfe des Extension-Mechanismus um neue Funktionalitäten zu erweitern. Jede Komponente kann dabei einen individuellen Teil zur graphischen Benutzungsoberfläche beisteuern, der nur zum Tragen kommt, wenn die Komponente tatsächlich in eine Anwendung integriert wird.

Die Workbench ist nach Überwinden einer steilen Lernkurve unserer Erfahrung nach gut geeignet, funktionsreiche und graphisch ansprechende Benutzungsoberflächen überdurchschnittlich schnell zu entwickeln. Diese sind in einem hohen Maße integriert und zugleich offen für Erweiterungen. Spezielle Oberflächenelemente, die dem Endbenutzer bei der Anpassung von Funktionalität helfen können, sind bereits vorhanden. Dazu zählt u. a. die Unterstützung für Konfigurationsdialoge und Präferenzeinstellungen (*Preferences*), das Filtern von Details der Benutzungsoberfläche in Abhängigkeit von Aktivitäten des Benutzers (*Activities*) und eine API für *Wizards*.

Wir haben gezeigt, dass sich die Eclipse Workbench und das Eclipse-Plugin-Modell mit seinem Erweiterungsmechanismus und der OSGi-konformen Runtime technisch eignen, um eine komponenten- und servicebasierte Architektur umzusetzen. Eine fachlich tiefe Integration der einzelnen Komponenten lässt sich dabei durch den Entwurf eines Kerns (Anwendungsmodell + fachliche Services) erreichen, der die RCP-Platform fachlich spezialisiert.

6 Zusammenfassung

In diesem Beitrag haben wir gezeigt, dass End-User Development zu flexibler und gut an die Benutzerbedürfnisse angepasster Software beitragen kann. Zur exemplarischen Verdeutlichung dieser Anpassungsbedürfnisse haben wir die an einen Leitstand-Arbeitsplatz gestellten Anforderungen beschrieben. Aus den Anforderungen ergeben sich Konsequenzen für die Architektur eines anpassbaren Leitstands. Konfigurierbarkeit und Erweiterbarkeit sind nicht ausschließlich an Interaktions- oder Oberflächenkonzepte geknüpft, sondern benötigen eine darauf ausgerichtete Softwarearchitektur. Mit den Konzepten *Komponente* und *Service* lässt sich eine geeignete Architektur für einen anpassbaren Software-Leitstand gestalten, die mit Hilfe der Eclipse Rich Client Platform umgesetzt werden kann. Ein Pilotprojekt am Deutschen Elektronen Synchrotron (DESY) soll die Tragfähigkeit der vorgeschlagenen Architektur unter Beweis stellen.

Danksagung. Dieser Beitrag ist im Rahmen des Forschungsprojekts „Component-based End User Development (CoEUD): Entwicklung und Umsetzung von durch den Endbenutzer dynamisch anpassbaren Infrastrukturen komponenten-basierter Systeme und Services für qualifizierte Arbeitsbereiche“ entstanden. Dieses Projekt wird vom Bundesministerium für Bildung und Forschung im Zuge der Forschungsoffensive "Software Engineering 2006" unter dem Kennzeichen 01ISF01B gefördert.

Literaturverzeichnis

[CPFM03] Costabile, M. F., Piccinno, A., Fogli, D., Mussio, P.: Software Shaping Workshops: Environments to Support End-User Development. In: Proceedings of the Workshop on End-User Development, Fort Lauderdale, 2003.

[Co06] Internetauftritt des Projekts “Component Based End User Development (CoEUD)”, http://www.coeud.org, 31.10.2006.

[CSS06] Internetauftritt des “Control System Studio” Projekts am Deutschen Elektronen Synchrotron (DESY), http://css.desy.de, 31.10.2006.

[DIN19222] Deutsches Institut für Normung e.V.: Vornorm DIN V 19222 – Leittechnik – Begriffe, Berlin: Beuth, 2001.

[Ec06] Eclipse Platform Whitepaper, http://www.eclipse.org/articles/Whitepaper-Platform-3.1/eclipse-platform-whitepaper.html, 31.10.2006.

[Eq06] Internetauftritt des Eclipse Equinox Projekts, http://www.eclipse.org/equinox/, 31.10.2006.

[Eu03] Internetauftritt des Projekts “Network of Excellence on End User Development”, http://giove.cnuce.cnr.it/eud-net.htm, 31.10.2006.

[GB04] Gamma, E., Beck, K.: Contributing to Eclipse – Principles, Patterns and Plug-Ins. Boston et al.: Addison-Wesley, 2004.

[HS05] Heiland, M., Schuhmacher A.: Hinter der Maske – Von OSGi-Features bei der Entwicklung von Plug-Ins profitieren, In: Eclipse Magazin, Volume 3, S. 65-73, 2005.

[LPKW06] Lieberman, H., Paternò, F., Klann, M., Wulf, V.: End-User Development: An Emerging Paradigm. In: Lieberman, H., Paternò, F., Wulf, V. (Hrsg.): End User Development. Dordrecht: Springer, 2006, S.1-8.

[LPRW03] Lieberman, H., Paternò, F., Reepenning, A., Wulf, V.: Proceedings of the Workshop on End-User Development, Fort Lauderdale, 2003.

[LPW06] Lieberman, H., Paternò, F., Wulf, V.: End User Development. Dordrecht: Springer, 2006.

[Kl03] Klann, M.: EUD-Net’s Roadmap to End-User Development. In: Proceedings of the Workshop on End-User Development, Fort Lauderdale, 2003.

[Me97] Meyer, B.: Object-oriented Software Construction. New Jersey: Prentice-Hall, 1997.

[Mø95] Mørch, A.: Three Levels of End-user Tailoring: Customization, Integration, and Extension. In: Proceedings of the 3rd Decennial Conference: Computer in Context: Joining Forces in Design, Aarhus, 1995.

[Mø04] Mørch, A., Stevens, G., Won, M. et al.: Component-Based Technologies for End-User Development. In: Communications of the ACM, Volume 47, Issue 9, Seite 59-62, 2004.

[OSGi06] Internetauftritt der OSGi Initiative, http://www.osgi.org, 31.10.2006.

[Pi05] Pipek, V.: From Tailoring to Appropriation Support: Negotiating Groupware Usage, Dissertation am Department of Information Processing Science, University of Oulu, 2005.

[St02] Stevens, G.: Komponentenbasierte Anpassbarkeit. Diplomarbeit am Institut Informatik, Rheinische Friedrich-Wilhelms-Universität Bonn, 2002.

[Sz98] Szyperski, C.: Component Software: Beyond Object-Oriented Programming. Massachusetts: Addison-Wesley, 1998.

[Zü05] Züllighoven, H.: Object-Oriented Construction Handbook. Heidelberg: dpunkt, 2005.