Susanne Robra-Bissantz, Bernhard Rumpe, Patrick Helmholz, Elena Kozlowski, Dirk Reiß, Silke Siegel

Software Engineering serviceorientierter Prozesse

In einer Zeit, in der Unternehmen zunehmend darauf angewiesen sind, dass ihre Stakeholder (z. B. Mitarbeiter, Kunden) Prozesse freiwillig und/oder jenseits traditioneller Unternehmensgrenzen vorantreiben, wird die Serviceorientierung zu einem Paradigma, das sich auf die Gestaltung und IT-Unterstützung der Prozesse selbst beziehen muss. Denn allein ein Prozess, der dem Mensch als Dienstleistung einen Nutzen verspricht, die Arbeit deutlich erleichtert und an seinen Präferenzen ausgerichtet ist, wird nachhaltig von diesem akzeptiert.

Prozessorientierte Informationssysteme können durch eine situationsspezifische, selbstadaptierende Software serviceorientiert und damit nutzergerechter gestaltet werden und so zu besserer Akzeptanz im Unternehmen und für dessen Partner führen. Dieser Artikel beschreibt die grundlegenden Konzepte der Situierung und skizziert, wie diese durch eine geeignete Softwarearchitektur realisiert werden kann.

Inhaltsübersicht

1 Akzeptanzprobleme von IT-Lösungen

Die IT-Unterstützung von betrieblichen Prozessen basiert heute zunehmend auf serviceorientierten Architekturen. Hier werden möglichst mehrfach nutzbare elektronische Services definiert und implementiert, die einzelne Aktivitäten im Prozess an vorher festgelegten Stellen unterstützen. Für die Softwareentwicklung im Unternehmen birgt dies eine Reihe von Vorteilen. Services sind wiederverwendbar und dabei flexibel zur Unterstützung verschiedener Geschäftsprozesse zusammenzustellen. Für den Nutzer prozessorientierter Anwendungssysteme stellt sich der IT-gestützte Prozess jedoch häufig als wenig flexibel dar. Stattdessen gibt er ihm, unabhängig von seinen Fähigkeiten, Präferenzen und aktuellen Problemen, relativ starr einen bestimmten Weg zur Aufgabenerfüllung vor.

Als Folge zeigt der Nutzer häufig eine nur mäßige Akzeptanz derartiger Systeme, er nutzt sie nicht im gewünschten Maße oder bricht die Bedienung ab. Besonders nachteilig ist dies, wenn das Unternehmen, wie z. B. in Shopping-Umgebungen oder im Wissensmanagement, darauf angewiesen ist, dass er die IT-Unterstützung mehr oder weniger freiwillig in Anspruch nimmt.

Eine potenziell höhere Akzeptanz entsteht, wenn sich die Software flexibel an den Benutzer anpasst und individuell auf diesen und seine aktuellen Bedürfnisse reagieren kann. Genau diese Kriterien werden erfüllt, wenn man die IT-gestützte Prozessführung selbst als Service betrachtet. Der Prozess übernimmt dabei die

[RBR+09] S. Robra-Bissantz, B. Rumpe, P.Helmholz, E. Kozlowski, D. Reiß, S. Siegel
Software Engineering serviceorientierter Prozesse
In: HMD - Praxis der Wirtschaftsinformatik, Heft 266, pp. 26-34, dpunkt.verlag, Heidelberg, April 2009
www.se-rwth.de/publications

Eigenschaften eines digitalen Service, wie sie aus der traditionellen Dienstleistungstheorie bekannt sind (z. B. [Bodendorf 1999]):

- Er integriert den Nutzer.
- Er reagiert individuell auf jeden einzelnen Nutzer und dessen Interaktionen.
- Er ist flexibel (und befindet sich damit in einem kontinuierlichen Verbesserungsprozess).

Neben der Forderung nach einer serviceorientierten Architektur zur Steigerung der Flexibilität in der Softwareentwicklung tritt damit die Forderung nach serviceorientierten Prozessen, die eine flexible, individuelle Unterstützung des IT-System-Nutzers gewährleisten.

In diesem Beitrag ist ein Ansatz zur Umsetzung serviceorientierter Prozesse dargestellt. Er basiert auf dem Konzept der Situierung zur Gestaltung des gesamten Informationssystems und leitet daraus passende Architekturstile und Programmierparadigmen des Software Engineering ab.

2 Situierung

Ein Ansatz zur serviceorientierten Gestaltung von Prozessen ist die Situierung. Sie bedeutet, allgemein begriffen, die Ausrichtung des Informationssystems an Situationen eines Geschäftsobjekts, z. B. des Menschen (Kunde, Mitarbeiter usw.), eines Dokuments oder einer betrieblichen Aufgabe. Die Situation ist ein relevanter Zustand des betrachteten Geschäftsobjekts, also ein Zustand, auf den die Software reagieren muss. Die Situierung geht damit über die Individualisierung hinaus [Meier et al. 2007] und betrachtet nicht allein langfristig stabile Eigenschaften des Geschäftsobjekts, sondern vielmehr auch dessen kurzfristige Zustände. Damit nutzt sie den Gedanken der Kontextorientierung aus dem Pervasive Computing (z. B. [Fleisch & Dierkes 2003]) auch in rein virtuellen Umgebungen.

Die Ziele der Situationsbetrachtung sind:

- Mitarbeiter oder sonstige Prozesskunden flexibel und individuell durch Prozesse zu steuern, ohne ihnen starre Aktivitäten vorzugeben;
- Risiken in der Prozessausführung zu erkennen, wie z. B. den drohenden Abbruch des Einkaufs durch den Kunden oder eine potenziell verspätete Erledigung einer Aufgabe;
- die Servicequalität zu verbessern, indem das System automatisiert auf spezielle Bedürfnisse oder Präferenzen des Prozesskunden reagiert.

Innerbetrieblich ist ein wesentliches Geschäftsobjekt der Mitarbeiter. Eine serviceorientierte Gestaltung und IT-Unterstützung von innerbetrieblichen Prozessen richtet sich damit an den Bedürfnissen, Präferenzen und/oder Problemen des Mitarbeiters aus. Die Software steuert ihn nicht starr durch gesamte Prozesse, sondern reagiert stattdessen auf seine Situationen und passt sich damit automatisch individuell an ihn an.

Fachlich konzeptionell schlägt sich der Ansatz der Situierung in einer genauen Betrachtung und Analyse der Situation eines Geschäftsobjekts, in den Aufgaben situierter Software sowie in ihrer Gestaltung nieder.

2.1 Situation eines Geschäftsobjekts

Die Situation eines Geschäftsobjekts, hier des Mitarbeiters, besteht aus (vgl. Abb. 1):

- einer beobachtbaren und beschreibbaren externen Situation, z. B. dem Klickverhalten und dem Inhalt einer Website;
- einer objektiven internen Situation, die nicht beobachtbar ist und beispielsweise seinen kognitiven und affektiven Zustand oder seine Pläne und Vorhaben umfasst; sie ist verhaltenswissenschaftlichen Erklärungen zugänglich, z. B. Ansätzen des Entscheidungsverhaltens;
- einer subjektiven internen Situation, die die objektive interne Situation eines Individuums

in seinen Prozessen konkretisiert; sie ergibt sich beispielsweise aus einer Verfeinerung der objektiven internen Situation mithilfe von vorhandenen Informationen über den Einzelnen (z. B. Kundenprofilen) aus vorhergehenden Beobachtungen des Individuums;
- den Anforderungen an das Informationssystem, die sich aus der subjektiven internen Situation ergeben, z. B. Hilfestellungen, Informationen oder Vorschläge für die weitere Vorgehensweise.

Die Situation einer Person ändert sich bei Aktivitäten ihres Umfelds oder durch eigene Aktivitäten. Dabei ist es, aufbauend auf theoretische Erkenntnisse zur menschlichen Interaktion, möglich, die interne Situation eines Menschen und damit sein Informationsbedürfnis aus seinen beobachtbaren Aktivitäten und Zuständen abzuleiten. Denn jegliche beobachtbare Aktivität eines Menschen ist das Resultat innerer Prozesse. Der Mensch nimmt seine externe Situation als Stimulus wahr, z. B. sein Umfeld oder die Aktionen eines Gesprächspartners genauso wie Umgebung und Interaktion eines Softwaresystems. Er reagiert darauf mit internen Aktivitäten, wie affektiven oder kognitiven Prozessen, beispielsweise während seiner Entscheidungsfindung. Zusammen mit längerfristigen individuellen Eigenschaften, beispielsweise seinen Präferenzen, resultiert eine extern beobachtbare Aktivität, z. B. der Aufruf einer weiteren Funktion. Diese wiederum verändert seine externe Situation.

2.2 Aufgaben in der Informationssystemgestaltung

Aus fachlicher Sicht basiert die Gestaltung von situierten Informationssystemen auf der Beschreibung möglicher externer und der Erklärung dazugehöriger interner Situationen des Geschäftsobjekts bzw. des Mitarbeiters. Hierzu stehen hinsichtlich seines beobachtbaren Interaktionsverhaltens einfache empirische Erkenntnisse zur Verfügung [Robra-Bissantz & Zabel 2005]. So deutet beispielsweise ein mehrfacher Aufruf derselben Webseite oder eine Navigation in »Schleifen« immer gleicher Seitenabfolgen eines einzelnen Bedieners auf seine Hilflosigkeit hin. Weiter gehend zeigen theoretische Erkenntnisse, z. B. aus der Usability von Anwendungssystemen, welche Hürden wo und aus

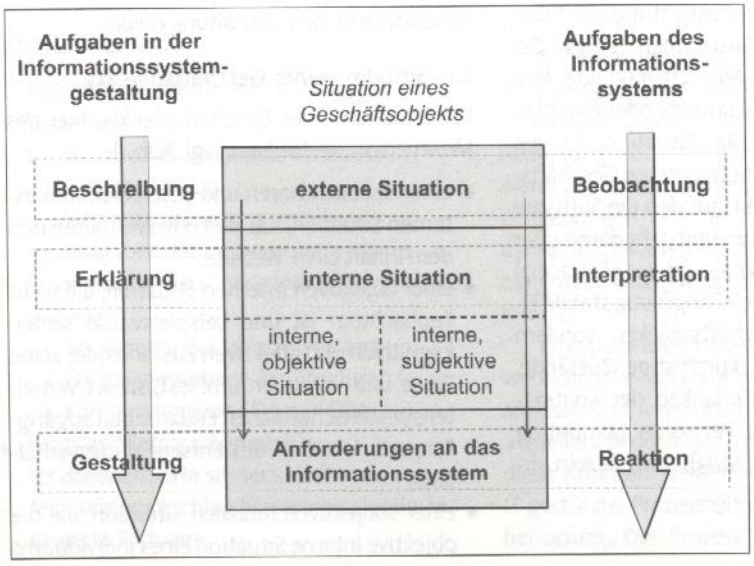

Abb. 1: Situation eines Geschäftsobjekts

welchem Grund bei der Bedienung innerbetrieblicher Softwaresysteme auftreten [Nielsen & Loranger 2008]. Es entsteht somit eine Sammlung von potenziellen Situationen des Mitarbeiters, die erkannt werden können und mit geeigneten Anforderungen an die Software in diesen Situationen hinterlegt sind.

2.3 Aufgaben des Informationssystems

Die Software selber hat in der Interaktion mit ihrem Nutzer die Aufgabe,

- seine beobachtbare Situation und damit z. B. seine Interaktion und sein Umfeld im Anwendungssystem zu erfassen,
- die beobachteten Daten hinsichtlich der internen Situation des Nutzers, beispielsweise mithilfe von Regelwerken oder dem Case-Based Reasoning, zu interpretieren
- sowie auf seine Situation zu reagieren und ihm z. B. die passende Information über Pop-up-Fenster oder Frames zu übermitteln.

Zur Beobachtung des Nutzers im Anwendungssystem stehen Ansätze der Logfile- und Clickstream-Analyse zur Verfügung. Sie offenbaren die Herkunft des Nutzers, Begriffe, die er in Suchfelder eingibt, die Reihenfolge aufgerufener Seiten und die Verweildauer auf ihnen sowie in ausgereiften Implementierungen seine Mausklicks und -bewegungen ebenso wie Tastatureingaben. Zusätzlich ist, beispielsweise über Skripte, erkennbar, in welcher Reihenfolge und Geschwindigkeit Formulare ausgefüllt werden.

Zunehmend attraktiv erscheint die Möglichkeit, über Sensoren, z. B. in der Maus, biometrische Daten des Nutzers zu ermitteln, die direkten Aufschluss über subjektive interne Zustände wie Emotionen oder das Involvement geben [Picard 2000].

Zu beachten ist allerdings, dass durch die Aufnahme und Analyse dieser persönlichen Daten eines Geschäftsobjekts die Persönlichkeitsrechte nicht verletzt werden [Eckert et al. 2007].

3 Architekturen für serviceorientierte situierte Software

Um die Situation eines Systemnutzers zu erfassen und entsprechend in Echtzeit auf diese reagieren zu können, ist eine adäquate Auswahl des Architekturstils aus bekannten Ansätzen [Buschmann et al. 1996] notwendig. Darin werden Entwurfsmuster [Gamma et al. 1995] wie das Observer-Muster (auch als Publish-Subscribe bekannt) geeignet kombiniert, um Effekte wie Komplexität und dynamische Anpassbarkeit zu erreichen. Die Nutzung solcher zunächst technischer Effekte für einen situierten Ansatz in der betrieblichen Software erlaubt die Übertragung dieser Fähigkeiten auf das Verhalten der Software ihrem Nutzer gegenüber.

3.1 Serviceorientierte Architekturen

Mit den serviceorientierten Architekturen (SOA) gibt es eine mittlerweile weit verbreitete technologische Grundlage, um moderne Unternehmensanwendungen in einzelne funktionale Komponenten zu zerlegen und damit die flexible Kombination einzelner Services zu einem gesamten, für den Nutzer bedienbaren Prozess zu ermöglichen. Zu beachten ist allerdings, dass SOA-Services zunächst als elektronische Funktionen für andere elektronische Funktionen konzipiert sind. Der Prozess als Service dagegen ist ein vom Menschen identifizierbarer Dienst. SOA-Services sind daher typischerweise feingranularer und nur in ihrer Komposition für einen serviceorientierten Prozess nutzbar.

3.2 Ereignisgetriebene Architekturen

Für die situationsspezifische Behandlung des beobachtbaren Nutzerverhaltens aus Klicks, Formulareingaben, Mausbewegungen und Ruhephasen bietet sich eine ereignisgetriebene Architektur (Event-Driven Architecture, EDA) an. Unter einem Ereignis ist in diesem Kontext ein signifikanter Zustandsübergang eines Systems zu verstehen, der sich technisch durch eine

Nachricht realisieren lässt [Schatten & Schiefer 2007].

Eine EDA zeichnet sich durch die Entkopplung der beteiligten Komponenten durch das Publisher-Subscriber-Muster aus. Komponenten können dabei Ereignisse verarbeiten und als Reaktion darauf weitere Ereignisse produzieren.

Über eine Messaging-Middleware auf Basis von CORBA oder dem moderneren .NET können sich Subscriber explizit bei anderen Komponenten anmelden, um über auftretende Ereignisse informiert zu werden, diese zu bewerten und ggf. darauf zu reagieren. Bei geschickter Kombination von Publishern und Subcribern entsteht eine Kette verarbeitender Einheiten, an deren Ende das Gesamtergebnis als Reaktion steht. Die Ereignisse können dabei sowohl von außen (extern) kommen (z. B. Benutzereingaben oder Messwerte) als auch vom System selbst (intern) ausgelöst werden (z. B. Änderungsbenachrichtigungen).

Um die Ereignisverarbeitung weiter zu flexibilisieren, bietet sich eine Veröffentlichung der erzeugten Ereignisse auch auf einer Eventbase gemäß dem Blackboard-Muster [Buschmann et al. 1996] an, die Ereignisse von allgemeinem Interesse jeder Komponente zugänglich macht. Ein Blackboard besitzt bezüglich der Ereignisverarbeitungskette ein gewisses »Gedächtnis« und kann so hervorragend als Aggregationsmechanismus eingesetzt werden, um die Historie des Nutzerverhaltens im Kontext bisher aufgetretener Ereignisse zu speichern. Aus diesen grundsätzlichen Architekturüberlegungen lassen sich einige Vorteile von EDAs ableiten:

- *Viele gleichzeitige Empfänger:* Die Systeme publizieren an einer Stelle, und der Publish-Subcribe-Mechanismus verteilt diese Nachrichten an alle interessierten Teilnehmer.
- *Aktualität:* Ereignisse werden versendet, sobald sie auftreten. Es entsteht sehr wenig lokale Latenzzeit, und auch ein iteriertes Polling und Busy-Wait ist nicht notwendig. Eine für den Nutzerkomfort notwendige »weiche« Echtzeit kann in ereignisverarbeitenden Ketten erreicht werden.
- *Asynchrone Verarbeitung:* Sender werden nicht durch Empfänger geblockt (wie im Beispiel der E-Mail).
- *Komplexe Ereignisse:* Beziehungen zwischen Ereignissen können durch spezielle Subscriber überwacht und der kausale Zusammenhang zwischen Ereignissen erkannt werden.
- *Wählbare Granularität:* Von den Systemen werden tendenziell eher spezifische Ereignisse verschickt als aggregierte Sammelereignisse [Hohpe 2006].

Die Granularität der Verarbeitungseinheiten und der damit im System auftretenden Ereignisse ist jedoch adäquat zu wählen, da sonst das Design unübersichtlich wird und die Fehlersuche einen erhöhten Aufwand erfordert [Hohpe 2006].

3.3 Kombination von EDA und SOA

Die Kombination von SOA-Komponenten mit dem ereignisorientierten Ansatz erlaubt die relativ modulare Entwicklung einer durch Nutzerereignisse induzierten, reaktiven Software. Insbesondere können beobachtende, bewertende und reaktive Komponenten des Systems relativ unabhängig voneinander realisiert, weiterentwickelt und auch später flexibel durch weitere ergänzt werden.

Je nach Auswahl der Granularität können Komponenten einer serviceorientierten Architektur und einer EDA identisch eingesetzt werden oder eine EDA-Komponente kann aus mehreren SOA-Diensten komponiert sein. SOA und EDA ergänzen sich insoweit, als SOA ähnliche Komponentenprinzipien nutzt, aber sie unterscheiden sich grundlegend im Kommunikationsprinzip, weshalb eine präzise, von der Architektur definierte Grenze beider Welten notwendig ist. Die Integration kann zum Beispiel durch ein Half-Sync-Half-Async-Architekturmuster [Schmidt et al. 2000] erreicht wer-

den. Im Zusammenspiel können die in einer serviceorientierten Architektur orchestrierten Dienste durch Ereignisse mit anderen Systemen verbunden werden (vgl. Abb. 2), was zu einer höheren Flexibilität führt [Rommelspacher 2008]. Gelegentlich wird die Verbindung der beiden Architekturen mit dem Begriff SOA 2.0 bezeichnet [Schatten & Schiefer 2007].

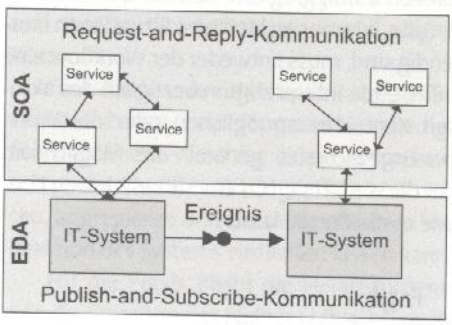

Abb. 2: Kombination von EDA und SOA (in Anlehnung an [Rommelspacher 2008, Bild 1])

Zusammengefasst kann die Kombination der EDA mit SOA-Komponenten Ereignisse aus unternehmensinternen und -externen Quellen einschließlich des Nutzerverhaltens schnell verarbeiten und bietet zugleich die notwendige Flexibilität, die einfache Änderungen und Rekonfigurationen von Prozessen erlaubt. Weiterhin ist eine adäquate Reaktion auch auf nicht antizipierte abnormale Ereignisse möglich [Rommelspacher 2008], weil der modulare Aufbau Robustheit in der Architektur und Flexibilität im Verhalten induziert.

4 Programmierparadigmen für serviceorientierte situierte Software

Der vorliegende Vorschlag einer Softwarearchitektur für situierte Anwendungssysteme basiert auf der Annahme, dass sie in einer konventionellen Programmiersprache wie Java oder C# realisiert werden soll. Im Zuge domänenspezifischer Sprachen gibt es jedoch auch Erweiterungen, die sich speziell für situierte Software eignen. Dazu gehört das Aspect-Oriented Programming (AOP) [Kiczales et al. 1997] und spezifischer noch das Context-Oriented Programming (COP) [Hirschfeld et al. 2008]. COP beispielsweise ergänzt eine Sprache um das Konzept des *Kontextes*. Ein Kontext kann den Zustand eines *Systems* und die aggregierte Historie des *Benutzerverhaltens* beinhalten und erlaubt so die direkte Programmierung von Reaktionen auf die Nutzersituation. COP bietet sogenannte *Layer*, die direkt im Quellcode codiert sind. Layer sind kontextabhängig »aktiviert« und werden dann ergänzend ausgeführt. Für die situationsabhängige Software werden so zusätzliche Verarbeitungsprozeduren für die auftretenden Ereignisse ermöglicht.

Eine disziplinierte Anwendung von Aspect-Oriented Programming erlaubt die Realisierung derselben Effekte durch Aspekte, die ebenfalls dem Basisverhalten situationsabhängig hinzugefügt werden können. Jedoch sind in AOP Kontexte selbst zu realisieren. Ein Vorteil ist allerdings, dass Basisverhalten und überlagerndes Zusatzverhalten besser entkoppelt sind und situationsspezifisches Zusatzverhalten damit besser weiterentwickelt werden kann.

5 Architekturvorschlag für serviceorientierte situierte Software

Ein nutzerspezifisch reagierendes System muss Komponenten beinhalten, die die externe Situation und damit den beobachtbaren Zustand sowie das Verhalten eines Nutzers *registrieren*, *interpretieren* und dementsprechend *reagieren*. So können etwa durch das Verfolgen des Klickverhaltens eines Nutzers Rückschlüsse darauf gezogen werden, wie gut dieser mit dem System zurechtkommt, und ihm im Bedarfsfall Hilfen angeboten werden. Wie beschrieben, bietet eine EDA die Möglichkeit, das Verhalten des Nutzers feingranular zu erfassen und durch Zuordnung zu einem Context-Layer flexible Anpassungen in der Reaktion durch Adaption

der Publisher-Subscriber-Ketten vorzunehmen. So wird der Benutzer mit den für ihn situationsabhängig am besten geeigneten Inhalten und/oder Diensten versorgt. Die Gesamtarchitektur eines solchen Systems ist in Abbildung 3 dargestellt.

Ein Nutzer interagiert mit dem klassisch in Webserver, Applikationsserver und Datenbank strukturierten System durch den Aufruf von Seiten, Navigation und das Ausfüllen von Formularen (a,d). Intern wird ein Model-View-Controller-Architekturmuster genutzt, um die Ablaufsteuerung und die Datenhaltung vom Applikationsserver als Funktionserbringer zu separieren (b,c). Jede Nutzeraktion wird als eigenständiges Ereignis gesehen und zur Erfassung des Nutzerverhaltens zusätzlich gespeichert. Dazu melden sich ein oder mehrere Verhaltensrecoder bei Webserver und Applikationsserver als Ereignis-Subscriber an, um sowohl Nutzerverhalten als auch Systemreaktionen (Fehlerrückfragen etc.) zu erfassen (e,f). Relevante Ereignisse werden von Verhaltensinterpretern aggregiert und interpretiert sowie im Ereignis-Repository für spätere Interpretationen gespeichert (j). Da hier wiederum ein Publish-Subscribe-Ansatz vorliegt (g,h), kann die Aggregation mit Verhaltensinterpretern auch mehrstufig kaskadiert werden. Über die Workflow-Engine in der Ablaufsteuerung kann der Kontext in die Reaktion des Systems einfließen (i), indem z. B. die geeigneten Services im Sinne einer SOA aufgerufen werden. Für eine Reaktion auf aggregierte Kontexte, wie sie z. B. im Falle interner subjektiver Situationen notwendig sind, muss entweder der Workflow eine umfassende Interpretation bezüglich des aktuellen Kontextes ermöglichen oder die Workflow-Engine bietet generell die Möglichkeit, über die vordefinierten Workflows hinweg Kontexte einfließen zu lassen.

6 Fazit

Um IT-gestützten Prozessen im Unternehmen und in der Kommunikation mit Unternehmenspartnern (z. B. Kunden) nachhaltig zu einer größeren Akzeptanz zu verhelfen, bietet es sich an, diese serviceorientiert zu gestalten. Im Sinne einer traditionellen Dienstleistung geht

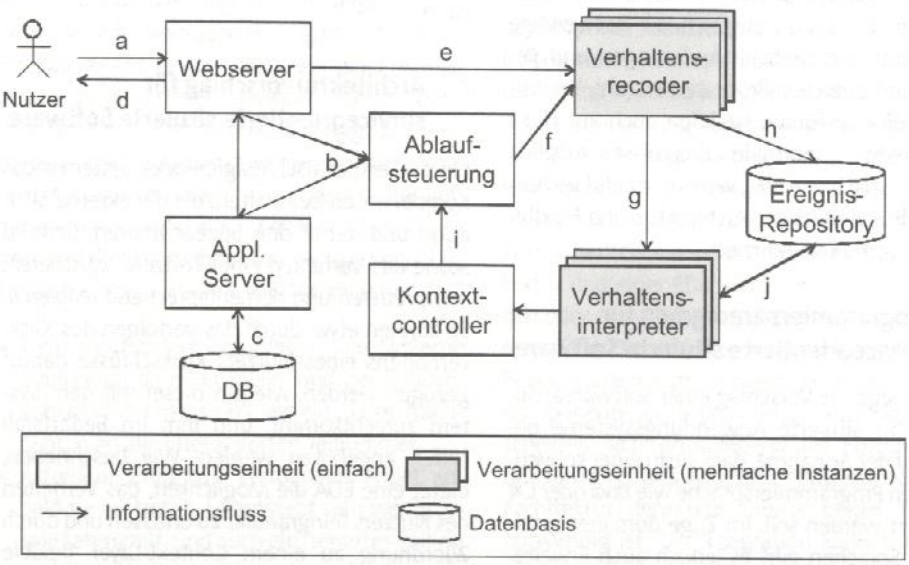

Abb. 3: Architekturvorschlag für situationsspezifisch agierende Softwaresysteme

die IT-Unterstützung bei serviceorientierten Prozessen interaktiv auf den Nutzer ein und reagiert flexibel auf seine Bedürfnisse, Probleme und Präferenzen.

Mit dem Basiskonzept der Situierung wird im vorliegenden Beitrag ein aus fachlicher Sicht neuer Weg zu serviceorientierten Prozessen vorgestellt. Er beruht darauf, dass die Software Situationen ihres Benutzers erkennt und flexibel und individuell darauf reagieren kann.

Eine Analyse verfügbarer Ansätze für Softwarearchitektur und Programmierparadigma führt zu einem Vorschlag einer Softwarearchitektur, die situationsabhängige Kontexte erfassen, aggregieren und historisieren sowie in die Reaktion des Systems einfließen lassen kann.

Für die Praxis bleibt die Herausforderung, wie die Situation des Nutzers hinsichtlich ihrer fachlichen Ausprägung und der Granularität des Kontextes identifiziert werden soll. Grundsätzlich bietet es sich bei der Neuentwicklung eines innerbetrieblichen Systems an, als ersten Kontext eine Einteilung in »Hilflos in Bedienung«, »Suche nach Prozessschritt« und »Wunsch nach schneller Bedienung« vorzunehmen. Speziell aus dem Bereich der IT-gestützten Prozesse in Webshops stehen daneben bereits erste inhaltliche Ansätze zur Verfügung [Robra-Bissantz & Zabel 2005]. Diese greifen z. B. typische Situationen eines Webeinkäufers, wie Hilflosigkeit, Ärger, Unsicherheit bezüglich des Bezahlungsverfahrens oder typische Schritte im Kaufverhalten, heraus und ermitteln theoretisch sowie in Experimenten hierzu passende beobachtbare Verhaltensweisen und Systemreaktionen.

Mit der vorgeschlagenen Softwarearchitektur ist es nun möglich, erste Situationen, Kontexte und Reaktionen durch flexibel ins System integrierbare Verhaltensrecoder und -interpreter sukzessive zu verfeinern. So können nutzerübergreifend statistische Auswertungen des Ereignis-Repositories zu einem lernenden System und damit adaptiven Verbesserungen führen, z. B. nach dem kollaborativen Motto »Viele Nutzer, die diesen Prozessschritt vorgenommen haben, haben auch ...«.

7 Literatur

[Bodendorf 1999] *Bodendorf, F.:* Wirtschaftsinformatik im Dienstleistungsbereich. Springer-Verlag, Berlin, 1999.

[Buschmann et al. 1996] *Buschmann, F.; Meunier, R.; Rohnert, H.; Sommerlad, P.; Stal, M.:* A system of patterns: Pattern-oriented software architecture: 1. Wiley & Sons, Chichester, 1996.

[Eckert et al. 2007] *Eckert, C.; Steinemann, B.; Wahl, T.:* SOA und Sicherheit: Web-Service Security Praxis und offene Probleme. In: Datenschutz und Datensicherheit, Jg. 31, 2007, Heft 9, S. 656-661.

[Fleisch & Dierkes 2003] *Fleisch, E.; Dierkes, M.:* Ubiquitous Computing aus betriebswirtschaftlicher Sicht. In: Wirtschaftsinformatik, Jg. 45, 2003, Heft 6, S. 611-620.

[Gamma et al. 1995] *Gamma, E.; Helm, R.; Johnson, R.; Vlissides, J.:* Design patterns. Elements of Reusable Object-Oriented Software. Addison-Wesley (Addison-Wesley Professional Computing Series), Reading, MA, 1995.

[Geihs 2008] *Geihs, K.:* Selbst-adaptive Software. In: Informatik-Spektrum, Jg. 31, 2008, Heft 2, S. 133-145.

[Hirschfeld et al. 2008] *Hirschfeld, R.; Constanza, P.; Nierstrasz, O.:* Context-oriented Programming. In: Journal of Object Technology, Jg. 7, 2008, Heft 3, S. 125-151.

[Hohpe 2006] *Hohpe, G.:* Programmieren ohne Stack: Ereignis-getriebene Architekturen. In: Objektspektrum, Jg. 13, 2006, Heft 2, S. 18-24.

[Kiczales et al. 1997] *Kiczales, G.; Lamping, J.; Mendhekar, A.; Maeda, C.; Lopes, C.; Loingtier, J.-M.; Irwin, J.:* Aspect-Oriented Programming. In: Aksit, M.; Matsuoka, S. (Hrsg.): ECOOP '97 – Object-oriented programming. Lecture Notes in Computer Science, 1241, Springer-Verlag, Berlin, 1997, S. 220-242.

[Meier et al. 2007] *Meier, M. C.; Winkler, V.; Buhl, H. U.:* Ansätze zur Gestaltung situierter und individualisierter Anwendungssysteme. In:

Wirtschaftsinformatik, Jg. 49, 2007, Sonderheft, S. 39-49.

[Nielsen & Loranger 2008] *Nielsen, J.; Loranger, H.:* Web Usability. Addison-Wesley, München, 2008 [Nachdr. der Ausg. 2006].

[Picard 2000] *Picard, R. W.:* Affective Computing. MIT Press, Cambridge, MA, 2000.

[Robra-Bissantz & Zabel 2005] *Robra-Bissantz, S.; Zabel, A.:* Push Concepts to Control Interactions in E-Commerce. In: CEC IEEE 2005 (Hrsg.): Seventh IEEE International Conference on E-Commerce-Technology. IEEE Computer Society Press, Los Alami, 2005, S. 168-175.

[Rommelspacher 2008] *Rommelspacher, J.:* Ereignisgetriebene Architekturen. In: Wirtschaftsinformatik, Jg. 50, 2008, Heft 4, S. 314-317.

[Schatten & Schiefer 2007] *Schatten, A.; Schiefer, J.:* Ereignisgetriebene Systeme verbessern SOA. In: iX – Magazin für professionelle Informationstechnik, Jg. 13, 2007, Heft 7, S. 122.

[Schmidt et al. 2000] *Schmidt, D. C.; Stal, M.; Rohnert, H.; Buschmann, F.:* Pattern-Oriented Software Architecture, Volume 2: Patterns for Concurrent and Distributed Objects. Wiley & Sons, Chichester, 2000.

Prof. Dr. Susanne Robra-Bissantz
Dipl.-Wirt.-Inf. Patrick Helmholz
Dipl.-Wirt.-Inf. Elena Kozlowski
Dipl.-Kff. Silke Siegel
Technische Universität Braunschweig
Institut für Wirtschaftsinformatik
Informationsmanagement
Mühlenpfordstr. 23
38106 Braunschweig
{s.robra-bissantz, p.helmholz, e.kozlowski, s.siegel}@tu-bs.de
www.tu-braunschweig.de/wi2

Prof. Dr. Bernhard Rumpe
RWTH Aachen
Software Engineering
Ahornstr. 55
52074 Aachen
rumpe@se.rwth-aachen.de
www.se-rwth.de

Dipl.-Wirt.-Inf. Dirk Reiß
Technische Universität Braunschweig
Institut für Software Systems Engineering
Mühlenpfordstr. 23
38106 Braunschweig
reiss@sse-tubs.de
www.sse-tubs.de