Muster und Cloud Computing als Plattformstrategie für mobile Unternehmenssoftware

Sebastian Damm, Thomas Ritz, Jakob Strauch

mobile media & communications lab
FH Aachen
Eupenerstr. 70
52066 Aachen
{s.damm,ritz,strauch}@fh-aachen.de

Abstract: Die Entwicklung mobiler Unternehmenssoftware wird durch einen stark variierenden Nutzungskontext und mannigfaltige Geräterestriktionen erschwert. Um den Anteil von Individualentwicklungen zu verringern, stellt dieser Beitrag ein Vorgehen vor, das auf Plattformstrategien sowie Lösungsmustern basiert. Die Muster dienen dabei insbesondere als plattformunabhängige Architekturkonzepte. Gerade diese Architekturmuster erlauben die Anwendung einer für den mobilen Einsatz geeigneten Abwandlung des Cloud Computings.

1 Motivation

Der Begriff Unternehmenssoftware hat sich als unscharfe Zusammenfassung von betrieblichen Anwendungssystemen etabliert. Die Einbindung von Mitarbeitern, die nicht an einem festen Standort arbeiten, ist das Ziel von mobiler Unternehmenssoftware [Ri03]. Die Anwendungen unterliegen im Gegensatz zu ihren Desktop-Pendants einem stetig variierenden Nutzungskontext (Netzverfügbarkeit, Lichtverhältnisse, Arbeitssituationen, etc.). Es gibt Ansätze und Vorgehensmodelle, um diese Besonderheiten in allen Phasen des Produktlebenszyklus möglichst vollständig zu erfassen und zu berücksichtigen [Ri07]. Insbesondere empfehlen sich benutzerzentrierten Vorgehen, die die Kunden und Anwender möglichst früh mit Prototypen konfrontieren. Dies ist vorwiegend in den Geschäftsfeldern wichtig, wo mobile Anwendungen Mitarbeiter adressieren, die bisher von der Büroautomatisierung weitestgehend unberührt geblieben sind. Ein früher Prototyp bietet eine gute Diskussionsgrundlage für die weitere Entwicklung. Der Einsatz mobiler Unternehmenssoftware ist für Unternehmen mit mobilen Mitarbeitern attraktiv. Mobile Lösungen können helfen vorhandene Geschäftsprozesse zu verbessern ([KPW03], [vV02]). Der ökonomische Erfolg hängt von der Ausnutzung der „Mobile added Values“[1] (kurz MAV) ab [PWT03]. Mobile Softwarelösungen bieten mehr Möglichkeiten als stationäre Lösungen. Informationen können *ubiquitär* und *kontextsensitiv* verarbeitet werden. Mit Sensoren und Aktoren ausgestattet kann die virtuelle Informationswelt mit der realen Umgebung *interagieren*.

[1] dt. *Mobile Mehrwerte*

Die Herausforderung besteht nun darin, Applikationen zu entwerfen, die die Besonderheiten bei der Entwicklung mobiler Unternehmenssoftware berücksichtigen. Zu diesen Besonderheiten zählen unter Anderen: Die Heterogenität und Restriktionen der mobilen Hardware (Smartphones, Notebooks und UMPCs) und deren Softwareplattformen, Netzverfügbarkeit, variierender Nutzungskontext (z.B. indoor/outdoor, Arbeitsumgebung) und Einschränkungen in der Usability (z.B. kleine Displays und Tasten). Dies führt zu Individualsoftware, welche für kleine und mittelständische Unternehmen schwieriger finanzierbar ist, als Standardsoftware. Die Anschaffung und der Betrieb einer IT Infrastruktur birgt zusätzliche Zeitaufwendungen und Investitionskosten (Netzwerke, Server, Lizenzen, etc.).

2 Verwandte Arbeiten

Im Folgenden werden ausgewählte Ansätze für die oben genannten Probleme vorgestellt. Es werden ihre Stärken und Schwächen im Kontext mobiler Unternehmenssoftware herausgestellt und erläutert, wie erkannte Schwächen durch Kombination der Ansätze kompensiert werden können.

2.1 Cloud Computing

Durch den stetigen Ausbau von Datennetzen und der Virtualisierungstechnik sind Betreibermodelle wie Cloud Computing möglich geworden. Zu dem Sammelbegriff „Cloud Computing“ existieren eine Vielzahl an Definitionen (u.a. [MG09], [Mi10a], [CH09], [Le09]). Es lassen sich jedoch einige grundlegende Merkmale festhalten. So kann man die Angebote grob in drei logische Schichten einteilen – **S**oftware-, **P**latform- und **I**nfrastructure **a**s **a** **S**ervice. Die Dienstleistungen werden von einem Anbieter für mehrere Kunden (mehrmandantenfähig) über das Web bereitgestellt. Die Abrechnung erfolgt in der Regel nutzungsbasiert.

Die Vorteile dieses Betreibermodelles liegen in der sehr kurzen „time-to-market“ sowie den geringen Investitionskosten, da die Infrastrukturen der Anbieter auf Virtualisierung basieren und lediglich angemietet werden. Somit bietet Cloud Computing eine Möglichkeit für KMU´s, Investitionen in IT Infrastruktur zu senken.

IaaS stellt die unterste Schicht im Cloud Computing dar. Es werden lediglich Rechen-, Netzwerk- und Speicherkapazitäten als Dienstleistung angeboten. Der Abstraktionsgrad ist somit sehr niedrig. Beispiel für ein derartiges Angebot ist Amazon´s S3 [Am10]. Mit PaaS bietet der Anbieter eine Plattform, um eigene Software „in der Cloud“, also auf der virtualisierten Infrastruktur, zu verwalten. Beispiel für PaaS ist Microsoft Azure [Mi10c], eine Plattform, um Webservices und Webanwendungen in der Cloud zu vertreiben. Bei SaaS werden begrenzt konfigurierbare Anwendungen angeboten, beispielsweise Standardanwendungen wie die CRM Lösung von salesforce [Sa10]. Die Kunden nutzen die Software über einen *Thin Client*, wie beispielsweise dem Webbrowser [Vo08]. SaaS bildet mit dem höchsten Abstraktionsgrad die oberste Schicht des Cloud Computing.

Für mobile Unternehmenssoftware ist dieses Softwaremodell (insbesondere SaaS) jedoch nur bedingt geeignet. Beispielsweise ist die Nutzung von Thin Clients nicht immer sinnvoll, da Netzverfügbarkeit und -qualität bei mobiler Software nicht garantiert werden kann. Software im geschäftlichen Umfeld sollte in „Offline Phasen" (zumindest eingeschränkt) bedienbar sein und ggf. auf Daten „adäquater Aktualität" arbeiten können [Ri03]. Darüber hinaus variieren mobile Browser in Performance, Javascript Funktionalität, RIA Unterstützung (Flash, Silverlight und Adobe Air) sowie Interaktion im Allgemeinen [He09], [Re09]. Des Weiteren bieten SaaS Anwendungen nur eingeschränkte Möglichkeiten zur Individualisierung, da die Anwendungen mit hoch standardisierten webbasierten Softwarekomponenten arbeiten, um so möglichst vielen Kunden gleichzeitig zur Verfügung zu stehen [Su08].

2.2 Muster

Muster (engl. Patterns) beschreiben ein *wiederkehrendes* Problem und definieren den *Kern* einer Lösung, sodass die Lösung an unterschiedliche Anwendungs-Kontexte angepasst werden kann. Man nutzt dazu eine semiformale Notation, um das bewährte Lösungswissen zu dokumentieren [Bu09]. Patterns werden nebst ausdrucksvollen *Namen*[2] mindestens durch die Beschreibung eines *Problems*, des zugrunde liegenden *Kontextes*, der *Lösung* sowie dessen *Konsequenzen* charakterisiert [BHS07b]. Muster ähnlichen Typs können zu Mustersprachen oder –systemen zusammengefasst werden. Die bekanntesten Mustersprachen sind in den Bereichen des Entwurfes von Softwarearchitekturen angesiedelt ([Ga08], [BHS07a]). Dennoch existieren auch weniger bekannte und nützliche Lösungsansätze in der Muster-Beschreibungsform, die Abschnitt 3 näher erläutert werden.

Da Muster bewährtes Lösungswissen enthalten, sind Sie gut geeignete Vorlagen für Softwarekomponenten. Aus den formalen Anteilen lassen Codefragmente ableiten. Aus den informellen Anteilen lässt sich die Anwendbarkeit im gegebenen Kontext begründen.

Muster sind nicht in ein ganzheitliches Vorgehensmodell integriert, sondern werden punktuell in der Entwicklung von Software eingesetzt. Die Wiederverwendung ist somit nicht organisiert.

2.3 Produktlinien

Das (Software) Produktlinien-Engineering (S-PLE) setzt auf „die organisierte *Wiederverwendung* und organisierte *Variabilität* auf Basis einer *gemeinsamen Plattform*" [BKP04]. Das Produktlinien Konzept hat seinen Ursprung in der klassischen Produktion von materiellen Gütern und wurde gegen Ende der 90er Jahre durch die Softwarebranche aufgegriffen. Das PLE teilt den Entwicklungsprozess in zwei Phasen: Domain- und Application-Engineering ([LRS07], [PBL05]).

[2] Teils metaphorisch, wie „Beobachter", „Erbauer" oder „Fassade", um die Kommunikation zu fördern

2.3.1 Domain Engineering

In der Phase des *Domain Engineering* werden typischerweise folgende Schritte durchlaufen

(D1) Eingrenzung der Zieldomäne
(D2) Festlegung der zu unterstützenden Plattform(en)
(D3) Entwicklung der Referenzarchitektur der Produktlinie
(D4) Ermittlung der zu unterstützenden Anwendungsfälle, Funktionen und deren Variationsmöglichkeiten

Die Variabilität findet sich in unterschiedlichen Artefakten wieder. Es lassen sich somit verschiedene Variabilitätstypen ableiten [PBL05], von denen hier lediglich ein Ausschnitt näher betrachtet wird:

(V1) Nicht-funktionale und qualitative Anforderungen
(V2) Funktionsumfang und Features
(V3) Prozesse und Aktivitäten
(V4) Datenumfang, -format und -zugriff
(V5) Benutzerschnittstelle

Die Variabilität kann durch sogenannte Variationspunkte beschrieben werden. Zu beachten ist, dass Variationspunkte in benachbarten Softwareschichten Einfluss auf einander haben (Datenhaltung←→ Geschäftslogik←→ Benutzerschnittstelle). Es gibt diverse formale Erweiterungen, um Variationspunkte in Entwicklungsartefakten explizit darzustellen ([KJD02], [vL02], [SP06], [BLP05]).

2.3.2 Application Engineering

Die anfänglichen Mehrkosten, die das Domain Engineering mit sich bringt, sollen durch Vorteile wie einer kürzeren „Time-to-Market“ aufgewogen werden [LRS07]. Diese Vorteile entstehen dadurch, dass in der Phase des *Application Engineering* die wesentlichen Bestandteile eines Softwareproduktes bereits durch die Produktlinie vorkonfiguriert und generiert werden können. Sowohl die Anwendungsfälle als auch die Varianten, die nicht von der Produktlinie abgedeckt werden, müssen individuell entwickelt werden. Die Erfahrungen und Ergebnisse solcher Individualprodukte können wieder in die Produktlinie zurückgeführt und diese somit für nachfolgende Produktableitungen genutzt werden. Zusammengefasst kann man das Application Engineering in folgenden Schritten darstellen:

(A1) Anforderungsanalyse (z.B. Soll-Prozesse)
(A2) Abgleich mit den in der Produktlinie bereits vorhandenen Anwendungsfällen
(A3) Auswahl der Varianten, die den kundenindividuellen Anforderungen am besten Entsprechen
(A4) Ggf. Implementierung der nicht abgedeckten Anforderungen
(A5) Ggf. Rückführung der Erfahrungen in die Produktlinie (Domain Engineering)

2.3.3 Vor- und Nachteile des S-PLE

Das Produktlinien-Engineering bietet ein gut strukturiertes Vorgehensmodell als langfristige Strategie. Durch die organisierte Wiederverwendung sind die Entwicklungsphasen für neue Produkte kürzer als bei einer individuellen Neuentwicklung. Nachteilig ist jedoch, dass die Zieldomäne und Referenzarchitektur in der Regel eng gefasst sind und es somit nicht möglich ist, die große Vielfalt an mobilen Geräten und Softwareplattformen mit einer einzigen Produktlinie abzudecken.

2.4 Forschungsfragen

Die vorherigen Abschnitte haben gezeigt, dass es derzeit keine ausreichend strukturierte Lösung gibt, mobile Unternehmenssoftware kundenindividuell und dennoch aus vorgefertigten Komponenten zu verlässlichen Kostenaussagen zu entwickeln.

Daher sollen die Vorteile der in Kapitel 2 vorgestellten Lösungsansätze zu einer integrierten Plattformstrategie zusammengefasst werden. Im Folgenden wird diese Strategie hergeleitet.

3 Herleitung der Plattformstrategie

Die Plattformstrategie nutzt das Vorgehensmodell aus dem *Produktlinien-Engineering*. Die Entwicklung wird demnach in die zwei Phasen Domain- und Application-Engineering eingeteilt. Während des Domain Engineerings werden Entwicklungsartefakte erstellt, die Variationspunkte enthalten. Als Entwurfsgrundlage sollen *Muster* mit hohem Abstraktionsgrad dienen, die (im Gegensatz zu reinen Vorlagen) den Anwendungskontext und die Konsequenzen semi-formal definieren. Im Application Engineering kann der Produktentwickler basierend auf diesen Mustern besser Entwurfsentscheidungen ableiten [DRS10]. Etwaige Services und mobile Komponenten, die anhand der Muster im Domain Engineering erstellt wurden, können ggf. wiederverwendet, konfiguriert und/oder angepasst werden. *Cloud Computing* ermöglicht eine kurze Bereitstellungszeit für die Service- und Backend-Infrastruktur, sodass in frühen Stadien der Entwicklung bereits prototypische Anwendungen gezeigt werden können. Die Prototypen dienen wiederum als Grundlage weiterer Verfeinerung und Individualisierung in Absprache mit den Kunden.

3.1 Domain Engineering

Die hier beschriebenen Aktivitäten orientieren sich an den in Abschnitt 2.1.1 definierten Schritten.

3.1.1 Zieldomäne (D1)

Als Zieldomäne dient der mittelständische Anlagen- und Maschinenbau. In dieser Domäne leisten mobile Servicetechniker Dienstleistungen wie Montage, Reparatur oder Wartung an Anlagen und Maschinen. Die anvisierten Produkte sollen typische Anwendungsfälle mit vorwiegend mobilem Anteil abdecken. Die mobilen Mitarbeiter arbeiten zeitweise an Orten mit eingeschränkter, teils nicht verfügbarer Konnektivität (Heizungskeller, Tagebau, etc.).

3.1.2 Plattform(en) (D2)

Ziel ist es, Komplettlösungen bestehend aus einer mobilen Applikation und einem serviceorientierten Backend anbieten zu können. Aufgrund der unterschiedlichen Ausprägungen von mobilen Anwendungsfällen, Lokationsbedingungen und Arbeitsumfelder sollen möglichst viele mobile Plattformen unterstützt werden.

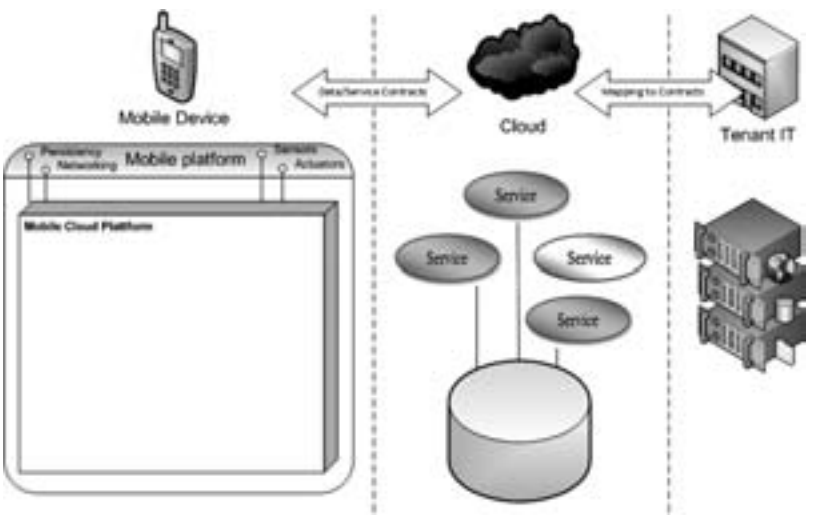

Abbildung 1 Architektur einer Kundeninstanz

Die mobile Basisarchitektur (Abbildung 1, linke Seite) setzt sich aus Entwurfs- und Architekturmustern zusammen (u.a. [Bi08], [BHS07a], [Fo08], [Mi09], [Al10]). Darüber hinaus soll durch Serviceorientierung eine Plattformunabhängigkeit realisiert werden [Jo08]. Die Daten und Teile der Geschäftslogik werden somit als Webservices implementiert. Es wurde festgelegt, dass die Webservice Architektur mit REST realisiert wird [Fi00], da REST-Services leichtgewichtig sind und mobile Geräte diese effizient nutzen können [HSA10]. Das Backend und die mobile Software kommunizieren lediglich mit diesen Webservices und nicht direkt mit Legacy Systemen (Datenbanken, etc.).

Der Produktentwickler soll entscheiden können, ob ein Netzausfall kompensiert werden kann. Um ein offline-fähigen mobilen Client realisieren zu können, wird das Offline Strategie Muster verwendet [RS10]. Der Entwurf des Musters ist so gestaltet, dass ein Client „nur online“, „nur offline“ oder „hybrid“ arbeiten kann. Durch die Anwendung des Musters ist es erforderlich, dass der Netzwerkstatus ermittelt werden kann und dass die mobile Plattform eine Persistenzschicht bietet (für Warteschlangen und Cache). Als Synchronisationsplattform dient dabei das Microsoft Sync Framework 4.0 (Oktober 2010 CTP) [Mi10a], das auf Client-Seite mehrere Plattformen unterstützt (iPhone, Silverlight, HTML5, Android, Windows Mobile/Phone 7). Serverseitig werden MS SQL Server oder das PaaS Angebot „Azure SQL“ von Microsoft unterstützt. (V1, V4)

Die Servicetechniker sollen in der Lage sein, die MAV´s auszunutzen, z.B. durch die Verwendung mobiler Sensoren (Scanner, GPS, etc.). Die zu unterstützenden Prozesse und der Funktionsumfang definieren somit die zu verwendeten mobilen Komponenten und Services in der Cloud. (V2 und V3)

Die Benutzerschnittstellen und Interaktionen sind abhängig von der Wahl des Endgerätes (z.B. Displaygröße), sowie dem Corporate Design des Kunden und der Zusammensetzung der Anwendungsfälle im Einzelfall. Hier sollte demnach wenig vorgegeben werden, um eine möglichst große Individualität zu gewährleisten (V5).

3.1.3 Referenzarchitektur (D3)

Zentraler Bestandteil ist eine Werkzeugpalette, mit deren Hilfe Muster und Vorlagen als auch Webservices und Komponenten in ein Repository kontinuierlich eingepflegt werden. Die Entwicklungswerkzeuge wurden vorwiegend mit dem „Visual Studio Visualization and Modeling SDK“ erstellt und können als Add-In´s in der Entwicklungsumgebung „Visual Studio 2010“ von Microsoft integriert werden [Mi10b]. Die Werkzeuge werden sowohl im Domain- als auch im Application Engineering eingesetzt, um Software für mobile Geräte und Webanwendungen in der Cloud erstellen und konfigurieren zu können. Jeder Kunde erhält somit eine Instanz einer auf ihn angepassten Softwarelösung.

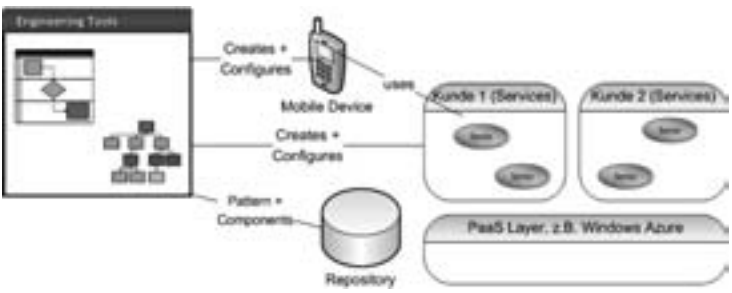

Abbildung 2 Architektur der Plattformstrategie

In der Instanz werden die Webservices (und Webanwendungen) gehostet. Die Services bieten Zugriff auf Daten (Basisdienste) und Geschäftslogik (Aktivitäten und Prozesse) an. Diese setzen (idealerweise) auf einer Platform as a Service Schicht auf, die der Software Anbieter selbst anmieten kann. Das Entwicklerteam des Anbieters ist somit in der Lage, schnell (virtuelle) Umgebungen für neue Kunden zu erzeugen, um möglichst früh eine prototypische (Gesamt-)Anwendung zeigen zu können.

3.1.4 Anwendungsfälle, Funktionen, Varianten (D4)

Die grundsätzlichen Prozesse in der Domäne ähneln sich. Jedoch sind einzelne Aktivitäten oder Subprozesse variabel. Die Auswahl der Varianten wird im Wesentlichen durch folgende Faktoren beeinflusst:

(F1) Lokations- und Arbeitsbedingungen
(F2) Verfügbare oder durch den Kunden gewünschte (mobile) Hardware
(F3) Wirtschaftliche Abwägungen (z.B. Verfügbarkeit vorhandener Komponenten für die gewählte Plattform)
(F4) Firmenrichtlinien
(F5) Vorhandene Infrastruktur

Um Variabilität in Prozessen zu unterstützen, werden *Prozessmuster* nach einem Ansatz von Hagen verwendet [Ha05]. Dazu werden Prozessmodelle mit dem Werkzeug „Mobile Process Modeler“ entworfen. Als Notation wird die BPMN 1.2 verwendet. Variable Prozessanteile werden mit Subprozessen abgebildet, die wiederum mit weiteren Prozessmodellen verknüpft werden. Zu den Subprozessvarianten muss jeweils festgelegt werden, unter welchen Rahmenbedingungen (F1 bis F5) sie einsetzbar sind. Diese Modelle können im Rahmen des Domain Engineerings aus dem Modeler in das Repository eingepflegt werden. Sie dienen zum Einen als Vorlage für die Konzeptionierung und Entwicklung der notwendigen Services und Komponenten für die mobile(n) Plattform(en). Zum Anderen werden die Prozessmuster im Rahmen der Anforderungsanalyse unter Berücksichtigung der beeinflussenden Faktoren F1 bis F5 (wieder)verwendet (Application Engineering, siehe Abschnitt 3.2). Die resultierenden Services und Komponenten werden ebenfalls im Repository abgelegt. (V2, V3)

Diese vorgefertigten Services und Komponenten verwenden ein Datenmodell, das ebenfalls Variationspunkte enthalten muss. Zu diesem Zweck werden sogenannte *Archetype Pattern* (Archetyp Muster) verwendet [AN04]. Arlow et al definieren einen (*Business) Archetype* als eine „ursprüngliche Entität, die konsistent und universell in Geschäftsdomänen auftritt“. Ein Archetype Pattern ist ein Zusammenspiel von Archetypes. Im Rahmen des Domain Engineerings werden insbesondere domänenspezifische Archetypes entworfen, wie sie in den Arbeiten von Piho et al zu finden sind [Pi10]. Diese Archetype Pattern werden ebenfalls im Repository angelegt. (V4)

Um die Benutzeroberfläche der mobilen Applikation an die individuellen Anforderungen und Anwendungsfälle anpassen zu können, wird das Model-View-Presenter Muster (kurz MVP) in der Variante „Passive View“ eingesetzt [Fo06]. Es entkoppelt die grafische Benutzeroberfläche von der Geschäftslogik und der Datenhaltung. Dadurch werden die Oberflächen leicht austauschbar. Das Repository enthält lediglich prototypische Oberflächen, die den Kunden einen ersten Eindruck von der Funktionalität der Software geben können. (V5)

3.2 Application Engineering

Bei der Erstellung eines konkreten Produktes werden die Soll-Prozesse analysiert, modelliert und auf ihr Mobilisierungspotential hin untersucht (A1). Dabei werden insbesondere die Lokationseigenschaften der Arbeitsumgebung (F1) mit Hilfe des „Mobile Location Profiles“ definiert [DRS10] und mit Prozessmodellen verknüpft. Anschließend wird anhand von Schlagworten im Repository nach bestehenden Vorlagen (Prozessmustern) gesucht (A2). Die Wahl der Prozessvarianten wird durch F1 bis F5 beeinflusst und orientiert sich an die definierten Rahmenbedingungen der jeweiligen Variante (A3).

Stellt sich heraus, dass passende Varianten noch nicht im Repository (für die anvisierte mobile Plattform) verfügbar sind, müssen diese individuell entwickelt werden (A4). Dem Produktlinien-Konzept folgend können Varianten mit Wiederverwendungspotential in das Repository zurückgeführt werden (A5).

4 Evaluation

In diesem Abschnitt wird die Tragfähigkeit der vorgestellten Plattformstrategie anhand einer prototypischen Entwicklung für einen typischen Anwendungsfall evaluiert. Dies entspricht der Phase D4 des Domain Engineering.

4.1 Anwendungsfall, Funktionen und Varianten

Ein Servicetechniker soll in die Lage versetzt werden, eine Anlage oder Maschine mit dem mobilen Gerät zu identifizieren, um nachgelagerte Prozesse (Maschinenhistorie einsehen, Wartungsbericht verfassen, etc.) durchführen zu können.

4.1.1 Prozessmuster erstellen

Das Identifizieren von Maschinen oder Anlagen kann über diverse Mechanismen unterstützt werden. Das Prozessmuster deutet die Variationspunkte mit dem Fragezeichensymbol an (Abbildung 3).

Das Modell wird mit weiteren Metadaten angereichert (Schlagworte, Titel, etc.). Schließlich können Prozess-Variationspunkte (V3) mit passenden Varianten verknüpft werden. Hier wird die Aktivität „Maschine identifizieren mit den Varianten „QR-Code (Kamera)“, „Service Tag“, „Kundenauswahl“ und „Barcode (Scanner)“ versehen.

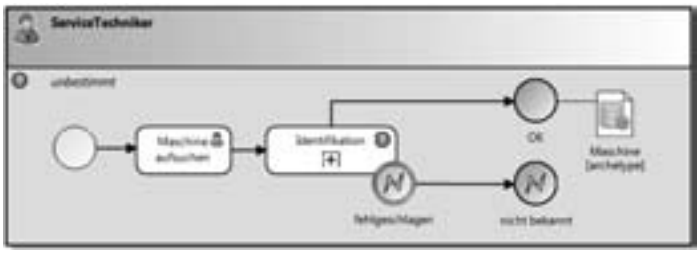

Abbildung 3 Teil des Prozessmusters "Maschine identifizieren"

Bei den Varianten handelt es sich wiederum um Prozessmodelle, die die Funktionsweise oder Interaktion darstellen. Dabei ist nicht entscheidend, dass die Modelle formal korrekt definiert werden, da kein Programmcode generiert werden soll. Vielmehr lassen sich unter Zuhilfenahme der Prozessmodelle Komponenten und Services erstellen, ins Repository hochladen und mit den Prozessmodellen verknüpfen. Abbildung 4 zeigt die Variante „QR-Code (Kamera)“, die auch die notwendigen Archetypes definiert. Desweiteren wird der Anwendungskontext unter Berücksichtigung der beeinflussenden Faktoren F1 bis F5 in den Metadaten des Modells (in Abb. nicht sichtbar) hinterlegt. So wird beispielsweise für die Variante „QR-Code (Kamera)“ festgehalten, dass diese in stark verschmutzten Umgebungen ungeeignet ist, da die QR Codes ggf. nicht lesbar sind (F1). Desweiteren ist entsprechende Hardware (Kamera) notwendig (F2).

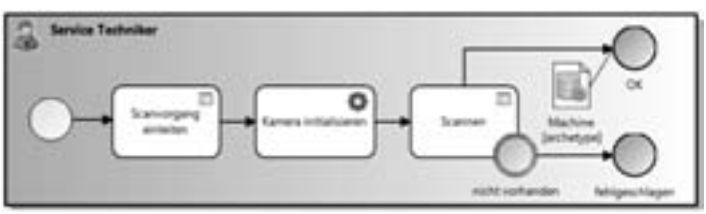

Abbildung 4 Prozessvariante "QR Code (Kamera)"

Aus dem Prozess und der Prozessvariante ist ersichtlich, welche Archetypen (Daten) verwendet werden. In diesem Fall ist es lediglich die Maschine.

4.1.2 Archetypen spezifizieren

Eine Maschine ist ein domänenspezifischer Archetyp, der aber durch die Spezialisierung des Basis Archetypen „Produktinstanz“ (*product instance archetype*) näher spezifiziert werden kann. Zu diesem Zweck wird „Machine“ mit dem „Business Archetype Modeller“ erstellt. Das Werkzeug kann das Modell in Codefragmente transformieren, die in den darauf basierten Komponenten und Services verwendet werden (Abbildung 5).

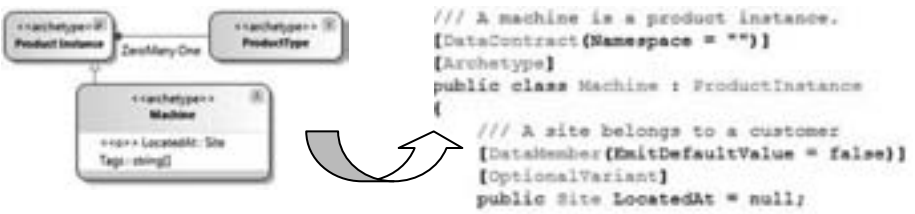

Abbildung 5 Archetype "Machine" und transformierter Datenkontrakt

4.1.3 Services und Komponenten

Anhand der Prozessmodelle und der Archetypen lassen sich mobile Komponenten und Webservices erstellen. Die Komponenten werden mittels MVP Muster implementiert. Die Schnittstellen für die grafische Benutzeroberfläche (V) werden durch die Komponenten (P) definiert und konsumieren die Webservices (M). Teile der Serviceschnittstellen können anhand der Archetypen definiert und generiert werden (Abbildung 5). Muster, Komponenten und Services werden im Repository für das Application Engineering vorgehalten.

4.2 Application Engineering

Während der Prozessanalyse bei einem Kunden, wurde festgestellt, dass in einem Prozess eine Maschine identifiziert werden muss. Der Entwickler prüft die Verfügbarkeit des Subprozesses im Repository und entscheidet anhand der Varianteninformationen und des kundenspezifischen Kontext, welche Variante gewählt werden kann (Abbildung 6).

Abbildung 6 Suche nach passenden Prozessen im Repository

Hat der Entwickler eine akzeptable Variante gefunden, so wird das entsprechende Prozessmodell aus dem Repository geladen. An diesem Modell sind sowohl die zu konfigurierenden Archetypen als auch die einzubindenden Komponenten (sofern vorhanden) annotiert.

5 Zusammenfassung und Ausblick

In diesem Beitrag wurde eine Plattformstrategie hergeleitet, die durch eine Kombination von *Produktlinien Engineering*, *Mustern* und *Cloud Computing* die kundenindividuelle Entwicklung mobiler Unternehmenssoftware aus wiederverwendbaren Artefakten ermöglicht. Anhand eines typischen Beispiels aus der Domäne „Service im Anlagen und Maschinenbau“ konnte das Zusammenspiel der entwickelten Werkzeuge, Muster und Komponenten als Plattformstrategie gezeigt werden.

Dazu diente das Produktlinien Engineering als grundlegendes Vorgehensmodell, um die Wiederverwendung und die Variabilität zu organisieren. Desweiteren werden Muster verwendet, die zentral in einem Online Repository als „Best Practice“ Vorlagen für Software Komponenten vorgehalten werden. Mit Hilfe des Cloud Computings kann ad hoc eine Service- und Datenhaltungs-Infrastruktur bereitgestellt werden, um so eine schnelle „Time-to-Market“ zu erreichen.

Die mobile Referenzarchitektur wird derzeit für Windows XP und höher, Windows Phone 7 sowie Windows Mobile 6.5 weiterentwickelt. Neben den bereits unterstützten Software Plattformen wird geprüft, ob Android und iOS als weitere Plattformen unterstützt werden können. Neben dem angeführten Beispiel werden derzeit weitere Anwendungsfälle (z.B. Maschinenhistorie oder Leistungserfassung) mit Hilfe der Plattformstrategie erstellt und weiterentwickelt. Eine ausführlichere Evaluation hinsichtlich der Wirtschaftlichkeit des Vorgehens kann erst nach Abschluss des Forschungsprojektes „Mobile Patterns as a Service“[3] mit den Projektpartnern vorgenommen werden.

Literaturverzeichnis

[Al10] Allamaraju, S.: RESTful web services cookbook. [Solutions for improving scalability and simplicity]. O'Reilly, Beijing, 2010.

[Am10] Amazon Simple Storage Service (Amazon S3). https://s3.amazonaws.com/.

[AN04] Arlow, J.; Neustadt, I.: Enterprise patterns and MDA. Building better software with archetype patterns and UML. Addison-Wesley, Boston, MA, 2004.

[BHS07a] Buschmann, F.; Henney, K.; Schmidt, D. C.: A pattern language for distributed computing. Wiley, Chichester, 2007a.

[BHS07b] Buschmann, F.; Henney, K.; Schmidt, D. C.: On patterns and pattern languages. Wiley, Chichester, 2007b.

[Bi08] Bishop, J.: C 3.0 design patterns. [use the power of C 3.0 to solve real-world problems]. O'Reilly, Beijing, 2008 erschienen 2007.

[BKP04] Böckle, G.; Knauber, P.; Pohl, K.: Software-Produktlinien. Methoden, Einführung und Praxis. dpunkt-Verl., Heidelberg, 2004.

[3] BMBF Förderprogramm „IngenieurNachwuchs“, Förderkennzeichen 17N1709

[BLP05] Buhne, S.; Lauenroth, K.; Pohl, K.: Modelling requirements variability across product lines. IEEE, 2005.

[Bu09] Buschmann, F.: A system of patterns. Wiley, Chichester, 2009.

[CH09] Catteddu, D.; Hogben, G.: Cloud Computing. Benefits, risks and recommendations for information security. http://www.enisa.europa.eu/act/rm/files/deliverables/cloud-computing-risk-assessment/at_download/fullReport, 01.12.2010.

[DRS10] Damm, S.; Ritz, T.; Strauch, J.: Benutzerzentrierte Anforderungsanalyse für die Produktlinien-Entwicklung mobiler Unternehmenssoftware. In (Klink, S.; Gesellschaft für Informatik / Fachgruppe EMISA 2010, K. Hrsg.). 07. - 08.10.2010 in Karlsruhe, Germany. Ges. für Informatik, Bonn, 2010.

[Fi00] Fielding, R. T.: REST: Architectural Styles and the Design of Network-based Software Architectures, 2000.

[Fo06] Fowler, M.: Passive View. http://martinfowler.com/eaaDev/PassiveScreen.html, 08.12.2010.

[Fo08] Fowler, M.: Patterns of enterprise application architecture. Addison-Wesley, Boston, Mass., 2008.

[Ga08] Gamma, E.: Design patterns. Elements of reusable object-oriented software. Addison-Wesley, Boston, 2008.

[Ha05] Hagen, M.: Definition einer Sprache zur Beschreibung von Prozessmustern zur Unterstützung agiler Softwareentwicklungsprozesse, 2005.

[He09] Hernandez, E. A.: War of the Mobile Browsers. In IEEE Pervasive Computing, 2009, 8; S. 82–85.

[HSA10] Hamad, H.; Saad, M.; Abed, R.: Performance Evaluation of RESTful Web Services for Mobile Devices. In (Arab Open University Hrsg.): International Arab Journal of e-Technology. Volume 1, No. 3, 2010; S. 72–78.

[Jo08] Josuttis, N. M.: SOA in practice. [the art of distributed system design]. O'Reilly, Beijing, 2008.

[KJD02] Kang, K.; Jaejoon Lee, P.; Donohoe: Feature-oriented product line engineering. In IEEE Software, 2002, 19; S. 58–65.

[KPW03] Kohdawandi, D.; Pousttchi, K.; Winnewisser, C.: Mobile Technologie braucht neue Geschäftsprozesse, 2003.

[Le09] Lenk, A.; Klems, M.; Nimis, J.; Tai, S.; Sandholm, T.: What´s inside the Cloud. An Architectural Map of the Cloud Landscape. https://wiki.gridx1.ca/twiki/pub/Main/VirtualizationProjectHome/An_Architectural_Map_of_the_Cloud_Landscape.PDF.

[LRS07] Linden, F.; Rommes, E.; Schmid, K.: Software product lines in action. The best industrial practice in product line engineering. Springer-Verlag Berlin Heidelberg, Berlin, Heidelberg, 2007.

[MG09] Mell, T.; Grance, P.: NIST Definition of Cloud Computing v15, 2009.

[Mi09] Microsoft Corp.: patterns & practices. Use Microsoft's proven practices for software engineering. http://msdn.microsoft.com/en-us/practices/default.aspx.

[Mi10a] Microsoft: Microsoft Sync Framework. http://msdn.microsoft.com/en-us/sync/default, 08.12.2010.

[Mi10b] Microsoft: Visual Studio Visualization and Modeling SDK. http://code.msdn.microsoft.com/vsvmsdk, 01.12.2010.

[Mi10c] Microsoft: Windows Azure. Microsofts´s Cloud Services Platform. http://www.microsoft.com/windowsazure/.

[PBL05] Pohl, K.; Böckle, G.; Linden, F.: Software product line engineering. Foundations, principles, and techniques ; with 10 tables. Springer, Berlin, 2005.

[Pi10] Piho, G. et al.: Towards Archetypes-Based Software Development. In (Sobh, T.; Elleithy, K. Hrsg.): Innovations in Computing Sciences and Software Engineering. Springer Science+Business Media B.V, Dordrecht, 2010; S. 561–566.

[PWT03] Pousttchi, K.; Weizmann, M.; Turowski, K.: Added Value-based Approach to Analyze Electronic Commerce and Mobile Commerce Business Models, 2003.

[Re09] Reynolds, F.: Web 2.0-In Your Hand. In IEEE Pervasive Computing, 2009, 8; S. 86–88.

[Ri03] Ritz, T.: Mobile CRM Systeme. In ZWF-Zeitschrift für wirtschaftlichen Fabrikbetrieb, 2003, 2003; S. 699–702.

[Ri07] Ritz, T.: Die benutzerzentrierte Entwicklung mobiler Unternehmenssoftware. In (Gesellschaft für Informatik Hrsg.): MMS 2007: Mobilität und mobile Informationssysteme. 2nd conference of GI-Fachgruppe MMS, 2007.

[RS10] Ritz, T.; Strauch, J.: "Offline Strategie"-Patterns für mobile SOA Prozesse. In (Bick, M.; Eulgem, S. Hrsg.): Mobile und ubiquitäre Informationssysteme - Proceedings zur 5. Konferenz MMS 2010, 23. - 25. Februar 2010 in Göttingen, Germany ; [im Rahmen der MKWI]. Ges. für Informatik, Bonn, 2010; S. 174–180.

[Sa10] Salesforce: CRM Software & Online CRM System. http://www.salesforce.com, 01.12.2010.

[SP06] Schnieders, A.; Puhlmann, F.: Variability Mechanisms in E-Business Process Families. In (Abramowicz, W.; Mayr, H. Hrsg.): 9th International Conference on Business Information Systems. BIS 2006, Klagenfurt, Austria, 2006.

[Su08] Sun, W. et al.: Software as a Service: Configuration and Customization Perspectives. IEEE, 2008.

[vL02] van der Maßen, T.; Lichter, H.: Modeling Variability by UML Use Case Diagrams. In (Avaya labs Hrsg.): International Workshop on Requirements Engineering for Product Lines, 2002; S. 19–26.

[Vo08] Vogel, O. et al.: Software-Architektur. Grundlagen - Konzepte - Praxis. Spektrum Akademischer Verlag, Heidelberg, Neckar, 2008.

[vV02] van der Heijden, H.; Valiente, P.: Mobile Business Processes: Cases from Sweden and the Netherlands, 2002.