Dienstleistungsmodularisierung zur kundenindividuellen Konfiguration

Martin Böttcher, Michael Becker, Stephan Klingner

Abteilung Betriebliche Informationssysteme
Institut für Informatik
Universität Leipzig
Johannisgasse 26
04229 Leipzig
boettcher@informatik.uni-leipzig.de
mbecker@informatik.uni-leipzig.de
klingner@informatik.uni-leipzig.de

Abstract: Die Dienstleistungswirtschaft ist zunehmend geprägt von dem Wunsch kundenindividuelle Lösungen bei gleichzeitiger Standardisierung der angebotenen Leistung zu ermöglichen. Ein hierfür geeigneter Ansatz ist die Modularisierung von Dienstleistungen. Abgeschlossene Dienstleistungskomponenten, welche für sich hochstandardisiert sein können, werden von Kunden zu einem individuellen Gesamtangebot zusammengesetzt. Um dies zu ermöglichen, bedarf es der Spezifikation von Dienstleistungskomponenten, der Definition ihrer Zusammenstellung sowie der Festlegung konfigurationsspezifischer Restriktionen und Regeln. Im vorliegenden Beitrag wird ein Metamodell vorgestellt, welches die Definition von Dienstleistungskomponenten, deren hierarchischen Abhängigkeiten und weiterer Konfigurationsregeln ermöglicht. Darauf aufbauend lassen sich kundenseitig individuelle Konfigurationen vornehmen.

1 Einführung

In der Dienstleistungswirtschaft gewinnen Business-to-Business-Dienstleistungen zunehmend an Bedeutung [Lo04]. Solche Dienstleistungen besitzen unter anderem eine hohe Komplexität, eine Vielzahl von Input- und Outputfaktoren sowie mehrere verantwortliche Geschäftseinheiten bzw. Personen. Beispiele hierfür sind produktnahe Installations-, Wartungs- und Reparaturdienstleistungen im Maschinenbau [Do03] sowie Full-Service-Leistungen im IT-Umfeld [BSF11]. Des Weiteren ist die Dienstleistungswirtschaft vergleichbar zu anderen Wirtschaftsbereichen vom Spannungsfeld der Standardisierung und Individualisierung geprägt. So sollen Dienstleistungen möglichst standardisiert werden, um Skaleneffekte zu erzielen und somit Kosten zu senken [FL97][PS00]. Zugleich verlangen Kunden individuelle Lösungen, die exakt ihren Anforderungen genügen. Um diesem Spannungsfeld zu begegnen, existiert im Produktionsbereich der Ansatz des Mass Customization [Pi99]. Hierbei werden die Vorzüge individueller und standardisierter Fertigung verbunden [DBF01][SB02].

erschienen im Tagungsband der INFORMATIK 2011
Lecture Notes in Informatics, Band P192
ISBN 978-3-88579-286-4
weitere Artikel online:
http://informatik2011.de/519.html

Um der Anforderung der gleichzeitigen Standardisierung und Individualisierung von Dienstleistungen gerecht zu werden, kann der Ansatz der Dienstleistungsmodularisierung Verwendung finden. Die hierbei definierten Dienstleistungskomponenten können standardisiert und zu individuellen Kundenangeboten zusammengefügt werden [BK10]. Dieser Modularisierungsansatz wird zunehmend für die Dienstleistungsdomäne diskutiert [AS10] und erste Ideen zu den Begriffen „Dienstleistungsmodul“, „Dienstleistungsbaustein“ und „Dienstleistungskomponente“ sind bereits dokumentiert [Bö04][CDG06][Em05][Bu07]. Viele dieser Arbeiten beschäftigen sich zunächst mit der grundlegenden Idee der Modularisierung von Dienstleistungen und definieren für die einzelnen Komponenten umfangreiche Attribute, um jede Dienstleistungskomponente für sich detailliert beschreiben zu können [O’S06][Be09]. Arbeiten, die in diesem Zusammenhang die Konfiguration betrachten, definieren die Varianz der Dienstleistungen über die Änderung der Attribute. Darüber hinaus ist es aber ebenso notwendig, die Kombination der einzelnen Dienstleistungskomponenten als wohldefinierte Konfiguration und somit die Zusammenstellung von Komponenten zu übergeordneten Komponenten, die letztendlich dann das Leistungsangebot umfassen, zu ermöglichen. Im Gegensatz zu den Komponenten der Produktionswirtschaft, bei denen die physische Abgrenzung im Vordergrund steht, muss bei Dienstleistungskomponenten die prozessuale und organisatorische Abgrenzung integriert und zeitgleich betrachtet werden [Bu02]. Die Besonderheit von Dienstleistungskomponenten ist somit darin zu sehen, dass eine jede Komponente letztendlich einen Prozessbestandteil widerspiegelt. Dieser Eigenschaft ist im Rahmen der Konfiguration bzw. der Darlegung der Konfigurationsmöglichkeiten Rechnung zu tragen.

In dem nachfolgend vorgestellten Metamodell wird ein Konfigurationsgraph definiert, welcher die Kombination von Dienstleistungskomponenten zu Gesamtleistungsangeboten ermöglicht. In einem zweiten Schritt wird dargelegt, wie die Attribute der einzelnen Komponenten auf vergleichbare Weise konfiguriert werden können. Diese Attribute ermöglichen die nähere Spezifikation einzelner Dienstleistungskomponenten. Zu den variablen Attributen, die im Rahmen der Konfiguration zu wählen sind, zählen hauptsächlich nicht-funktionale Attribute wie bspw. Ort, Zeit, Preis etc. Für die Definition des Konfigurations-Metamodells werden einerseits die existierenden Arbeiten der Dienstleistungsdomäne herangezogen und andererseits auf die Erfahrungen der Domänen der Produktionswirtschaft [St65][BC97][DBF01] und Softwaretechnik [Sa97][Sz02] zurückgegriffen. So wurde für die Produktionswirtschaft beispielsweise der Ansatz des Gozinto-Graphen entwickelt [Va62] und zu einem „und-oder-Graphen“ [BO03] weiterentwickelt. Für die Konfiguration innerhalb der Softwaretechnik existiert unter anderem der Ansatz der Feature-Modellierung [Ka90][CE00].

Für die Spezifikation von Modellen bedarf es einer zugrundeliegenden Modellierungstheorie. Diese erlaubt die präzise Darlegung, welche Art von Modell erstellt werden soll. Im vorliegenden Fall wird ein Metamodell erstellt, welches die Konzepte bereithält, um Modelle von Konfigurationsgraphen für die Dienstleistungskonfiguration erstellen zu können. Die Nutzung von Metamodellen zur Erstellung von Modellen entspricht den Arbeiten von [Fa05a][Fa05b][Kü04] und wurde für die Anwendung in der Dienstleistungsdomäne bereits von [Ke09] beschrieben.

Nachfolgend wird das Konfigurations-Metamodell, welches die Zusammenstellung von Leistungsangeboten aus Dienstleistungskomponenten ermöglicht, sowie eine Erweiterung um die Attributspezifikation und -konfiguration beschrieben. Die Erkenntnisse, die aus dem Metamodell gewonnen werden können, sollen abschließend diskutiert und weitere Forschungsfragen aufgestellt werden.

2 Konfigurations-Metamodell

Die Erstellung von Metamodellen erfolgt in einem sogenannten Technikraum, bei welchem die Meta-Ebenen sowie die einzusetzende Technologie definiert sind [Ke09]. Zu solchen Technikräumen gehören beispielsweise Architektur integrierter Informationssysteme [Sc02], Eclipse Modelling Framework [Bu04], Extensible Markup Language [Ha05] sowie Web Ontology Language [Hi08]. Um eine möglichst breite Anwendung des Konfigurations-Metamodells zu ermöglichen, wird zunächst auf die Festlegung auf einen Technikraum i.e.S. verzichtet und anstelle dessen die Spezifikation des Metamodells unter Zuhilfenahme mathematischer und aussagenlogischer Ausdrücke vorgenommen. In einem weiteren Schritt kann diese Spezifikation dann in einen konkreten Technikraum überführt werden.

2.1 Komponentenspezifikation

Wenngleich die Dienstleistungskomponenten mit umfangreichen Attributen zur Festlegung der funktionalen und nicht-funktionalen Eigenschaften [O'S06] versehen werden können, werden zunächst die Spezifikation eines Konfigurationsgraphen und die Zusammenstellung von Komponenten und nicht die Beschreibung der einzelnen Komponenten betrachtet. Es wird darüber hinaus innerhalb der Struktur des Konfigurationsgraphen eine Möglichkeit zur Konfiguration von Attributen vorgestellt. Die Menge der Dienstleistungskomponenten *Components* wird definiert als:

$$Components = \{component_1, \dots, component_n\}, n \in \mathbb{N}$$

Für die eindeutige Identifikation der einzelnen Komponenten wird diesen ein entsprechender Identifikator zugewiesen. Dies erfolgt über die Abbildung *ServiceIdentity* und die Menge *Identities*, welche alle verfügbaren Identifikatoren enthält:

$$ServiceIdentity: Components \rightarrow Identities$$
$$\forall c_1, c_2 \in Components: c_1 \neq c_2 \leftrightarrow ServiceIdentity(c_1) \neq ServiceIdentity(c_2)$$

Die Komponenten repräsentieren einzelne Bestandteile des Gesamtangebotes eines Unternehmens und werden in weiteren Schritten zueinander in Beziehung gesetzt. Diese Beziehungen besagen, wie sich eine Komponente aus anderen Komponenten zusammensetzt (Komposition) oder wie eine Komponente in weitere Komponenten zerlegt werden kann (Dekomposition).

erschienen im Tagungsband der INFORMATIK 2011
Lecture Notes in Informatics, Band P192
ISBN 978-3-88579-286-4
weitere Artikel online:
http://informatik2011.de/519.html

2.2 Hierarchischer Konfigurationsgraph

Die Zusammensetzung von Dienstleistungskomponenten bedeutet, dass eine zusammengesetzte Komponente einen Prozess repräsentiert, welcher die Summe der Prozessteile der Kindskomponenten ist. Für die Konfiguration ist es nun notwendig, darzulegen, wie sich eine Komponente aus anderen Komponenten zusammensetzen kann. In diesem Zusammenhang sind auch die möglichen Varianten der Zusammensetzung zu spezifizieren, um dem Kunden die Auswahl verschiedener Optionen zu erlauben.

Für die formale Beschreibung der Zusammensetzung der Komponenten wird zunächst ein gerichteter, zyklusfreier Graph, bestehend aus Kanten *E* (Edges) und Knoten *V* (Vertices) definiert:

$$Graph = (V, E)$$

$$E = V \times V, \forall (v_1, v_2) \in E: v_1 \neq v_2$$

Um die Auswahlmöglichkeit bei der Zusammensetzung von Komponenten zu ermöglichen, wird der Graph mit zusätzlicher Semantik angereichert, indem zwei verschiedene Knotentypen definiert werden. Damit geht der Graph über reine Zusammensetzungsgraphen wie bspw. dem Gozinto-Graphen hinaus. Dazu wird zusätzlich zur Menge der Komponenten *Components* eine weitere Menge *Connectors*, die sogenannte Konnektoren enthält, definiert:

$$V \supseteq Components \cup Connectors$$

Mithilfe eines Konnektors können Aussagen getroffen werden, dass eine bestimmte Anzahl der über ihn verbundenen Kindsknoten eines Elternknotens im Rahmen der Konfiguration zu wählen ist, bzw. dass die Kindsknoten als obligatorisch oder fakultativ anzusehen sind.

Es existieren zusätzliche Restriktionen, denen der erstellte Konfigurationsgraph genügen muss. Zunächst ist die Zyklenfreiheit zu gewährleisten, da sich eine Komponente, auch über mehrere Ebenen, nicht selbst enthalten darf:

$$\forall (v_1, v_2 \in V): \exists v_1 P v_2 \Rightarrow \nexists v_1 Q v_2, mit\ P, Q\ als\ Pfade\ aus\ G$$

Des Weiteren dürfen Komponenten innerhalb des Graphen nicht direkt miteinander verbunden werden, sondern es muss mindestens ein Konnektor zwischen ihnen definiert werden. Anderenfalls lassen sich keine konsistenten Aussagen zu den Regeln der Konfiguration spezifizieren:

$$\forall (v_1, v_2) \in E: \{v_1, v_2\} \nsubseteq Components$$

ISBN 978-3-88579-286-4

2.3 Kardinalitäten

Mit dem definierten Graphen ist die generelle Struktur spezifiziert worden, welche die einzelnen Komponenten sowie die Konnektoren enthält. Die Konnektoren können nun durch Kardinalitäten angereichert werden, mit welchen spezifiziert wird, wie viele der nachfolgenden Knoten während der Konfiguration gewählt werden müssen bzw. können. Hierbei wird eine Tupelmenge *{(min, max)}* definiert:

$$Cardinalities: Connectors \rightarrow \wp(\mathbb{N} \times \mathbb{N}), mit\ \wp\ als\ Potenzmenge$$

$$Cardinalities(c) = \{(min_1, max_1), \dots, (min_n, max_n)\}, n \in \mathbb{N}$$

$$\forall (min, max) \in Cardinalities(c): (min \leq max)$$

Bei diesen Tupeln legt *min* fest, wie viele der Kindsknoten bei der Konfiguration mindestens gewählt werden müssen und *max* spezifiziert, wie viele der Kindsknoten maximal gewählt werden dürfen. Sollten die beiden Werte identisch sein, wird damit festgelegt, dass eine feste Anzahl von Kindsknoten zu wählen ist. Des Weiteren lässt sich als Empfehlung ein Default-Wert einer Kardinalität definieren, welcher innerhalb des Wertebereiches eines Tupels liegen muss:

$$DefaultCardinality: Connectors \rightarrow \mathbb{N} \times \mathbb{N}$$

$$\forall (m, n) \in DefaultCardinality(c \in Connectors): m = n$$

$$\forall (n, n) \in DefaultCardinality(c): \exists (o, p) \in Cardinalities(c): o \leq n \leq p$$

Für die Kardinalitäten existieren unterschiedliche Restriktionen, die bei ihrer Spezifikation berücksichtigt werden müssen. Für diese Restriktionen wird zunächst eine Hilfsfunktion *postnodes* spezifiziert, welche die Kindsknoten eines Knoten identifiziert:

$$postnodes: V \rightarrow \wp(V)$$

$$postnodes(v) = \{v' \epsilon V: \exists e \in E: e = (v, v')$$

Nun lässt sich festlegen, dass der Maximalwert der Kardinalitäten nicht größer sein darf als die Anzahl der diesem Konnektor nachfolgenden Kindsknoten:

$$\forall c \in Connectors, \forall (m, n) \in Cardinalities(c): (n \leq |postnodes(c)|)$$

Des Weiteren sind überlappende Kardinalitäten redundant und könnten zu einer Kardinalität zusammengefasst werden. Demzufolge sind überlappende Kardinalitäten aufzulösen:

$$\forall c \in Connectors, \forall (m, n) \in Cardinalities(c):$$

$$\nexists (o, p) \in Cardinalities(c): (m \leq o \leq n) \vee (m \leq p \leq n)$$

2.4 Nicht-hierarchische Beziehungen

Die bisherige Graphenspezifikation erlaubt die Darlegung, dass eine Komponente sich aus einer Menge von Teilkomponenten zusammensetzt und aus diesen Teilkomponenten eine bestimmte Anzahl im Rahmen der Konfiguration zu wählen ist. Hierdurch können alle Abhängigkeiten innerhalb der Zusammensetzungshierarchie dargelegt werden. Im Rahmen einer Konfiguration können jedoch auch Abhängigkeiten zwischen einzelnen Komponenten existieren, die sich nicht innerhalb der Hierarchie abbilden lassen, da die Komponenten nicht explizit über einen Kind-Eltern-Pfad miteinander verbunden sind. Hierfür sind Definitionsmöglichkeiten für nicht-hierarchische Abhängigkeiten notwendig.

Die nicht-hierarchischen Abhängigkeiten können unter Zuhilfenahme aussagenlogischer Ausdrücke [Kr95] spezifiziert werden. Dadurch ist ein größtmöglicher Freiheitsgrad in der Spezifikation der Regeln gewährleistet. Zugleich lassen sich die erstellten Regeln bezüglich ihrer Konsistenz überprüfen. Allerdings bedarf es profunder Kenntnis um Formalien der aussagenlogischen Ausdrücke, um diese Regeln spezifizieren zu können. Um auch Fachabteilungen die Möglichkeit der Regelspezifikation geben zu können, wird ein Set vordefinierter Regeln bereitgestellt, welche aufgrund ihrer Bezeichnung allgemeinverständlich sind. Zugleich sind diese Regeln auf die Aussagenlogik abbildbar, um die Konsistenz zu weiteren aussagenlogisch spezifizierten Regeln zu gewährleisten und ihre Überprüfung zu erleichtern. Tabelle 1 definiert die Regeln, ihre aussagenlogische Darstellung sowie deren Beschreibung.

Regel	**Aussagenlogische Darstellung**	**Beschreibung**
a-führt-zu-b	$a \Rightarrow b$	Wenn die Komponente a gewählt wird, muss auch die Komponente b gewählt werden.
nicht-a-führt-zu-nicht-b	$\neg a \Rightarrow \neg b$	Wenn die Komponente a nicht gewählt wird, darf auch die Komponente b nicht gewählt werden.
a-und-b-schließen-sich-aus	$(a \Rightarrow \neg b)$ $\wedge (b \Rightarrow \neg a)$	Wenn die Komponente a gewählt wird, darf Komponente b nicht gewählt werden und umgekehrt.
nicht-a-führt-zu-b	$\neg a \Rightarrow b$	Wenn die Komponente a nicht gewählt wird, muss die Komponente b gewählt werden.

Tabelle 1: Nicht-hierarchische Abhängigkeiten

ISBN 978-3-88579-286-4

2.5 Temporale Abhängigkeiten

Dienstleistungskomponenten repräsentieren Prozesse und besitzen somit zeitliche Abhängigkeiten zueinander. Dies unterscheidet Dienstleistungskomponenten grundlegend von Produktkomponenten. Die Durchführungsreihenfolge bzw. die Abfolge der Dienstleistungskomponenten muss demzufolge im Konfigurationsgraphen berücksichtigt werden, damit nach der Konfiguration die korrekte Reihenfolge der ausgewählten Komponenten festgelegt ist. Aufgrund der umfangreichen Kombinationsmöglichkeiten im Rahmen der Konfiguration ist es sehr aufwendig, für alle denkbaren Kombinationen vorgefertigte prozedurale Prozessmodelle zu hinterlegen. Auch bei einer marginalen Änderung des Konfigurationsgraphen (bspw. durch das Herausnehmen einer Komponente) würden umfangreiche Änderungen der Prozessmodelle vollzogen werden müssen. Um die Flexibilität des Konfigurationsgraphen zu erhalten, können die zeitlichen Abhängigkeiten der einzelnen Komponenten nicht prozedural, sondern deklarativ erfolgen [AP06]. Hierbei werden nur die essentiellen zeitlichen Abhängigkeiten zwischen den Komponenten in Form von Regeln definiert. Das nach der Konfiguration zu erstellende deklarative Prozessmodell muss dann konform zu den deklarativen Regeln der ausgewählten Komponenten sein.

Deklarative Prozessdefinitionen greifen beispielsweise auf die Verwendung der Linearen Temporalen Logik (LTL) [DO00] zurück. Die LTL ist in ihrer Anwendung jedoch noch komplexer als die Aussagenlogik. Um die Anwendbarkeit der deklarativen Regeln zu gewährleisten, wird ähnlich zu den nicht-hierarchischen Abhängigkeiten eine Menge vordefinierter Regeln bereitgestellt. Diese sind einerseits allgemein verständlich, andererseits lassen sie sich auf die LTL abbilden, so dass die Vorzüge der LTL, wie bspw. die Anwendung weiterer Regeln oder die Regelprüfung verwendet werden können. Die Tabelle 2 zeigt die Menge der vordefinierten temporalen Regeln.

Die Anwendung der temporalen Regeln unterliegt Restriktionen, die sich aus der Struktur des Graphen ergeben. So dürfen temporale Abhängigkeiten nur zwischen Komponenten definiert werden, die nicht über einen direkten Pfad miteinander verbunden sind. Dies ergibt sich aus der Zusammensetzungslogik eines direkten Pfades, da der Prozess der Kindskomponente bereits in dem Prozess des Elternknoten enthalten sein wird bzw. sich der Prozess des Elternknoten aus den Prozessen seiner Kindsknoten ergibt. Neben den graph-strukturellen Restriktionen gibt es noch weitere Restriktionen, die sich aus den hierarchischen und nicht-hierarchischen Abhängigkeiten ergeben. So bedarf es keiner temporalen Abhängigkeit zwischen Komponenten, die sich gegenseitig aufgrund expliziter nicht-hierarchischer Abhängigkeiten oder durch die Maximalkardinalität eines Konnektors ausschließen.

erschienen im Tagungsband der INFORMATIK 2011
Lecture Notes in Informatics, Band P192
ISBN 978-3-88579-286-4
weitere Artikel online:
http://informatik2011.de/519.html

Regel	**LTL-Darstellung**	**Beschreibung**
a-benötigt-nachfolgend-b	$\Box(a \Rightarrow \diamond\, b)$	Wenn die Komponente *a* ausgeführt wird, muss irgendwann später die Komponente *b* ausgeführt werden.
a-benötigt-sofort-nachfolgend-b	$\Box(a \Rightarrow (\circ\, a \vee \circ\, b))$	Sofort nachdem Komponente *a* ausgeführt wurde muss Komponente *b* ausgeführt werden (oder nochmals Komponente *a*).
a-benötigt-vorhergehend-b	$\neg(\neg b U a)$	Bevor Komponente *a* ausgeführt werden kann, muss Komponente *b* ausgeführt werden. Die Definition der temporalen Logik legt hierbei fest, dass die Komponente *a* solange nicht ausgeführt werden darf, wie *b* nicht ausgeführt wurde.
a-benötigt-sofort-vorhergehend-b	$\neg \diamond (\neg b \wedge \circ\, a)$	Direkt vor der Ausführung der Komponente *a* muss die Komponente *b* ausgeführt werden. Hier definiert als solange *b* nicht ausgeführt wurde, darf *a* nicht ausgeführt werden.
a-darf-niemals-vor-b-ausgeführt-werden	$\neg \diamond (a \wedge \diamond\, b)$	Zu keinem Zeitpunkt darf die Komponente *b* ausgeführt werden, wenn die Komponente *a* bereits ausgeführt wurde.
a-darf-nicht-direkt-vor-b-ausgeführt-werden	$\neg \diamond (a \wedge \circ\, b)$	Zu keinem Zeitpunkt darf *b* direkt nach *a* ausgeführt werden.

Tabelle 2: Temporale Abhängigkeiten

ISBN 978-3-88579-286-4

2.6 Konfiguration von Attributen

Das spezifizierte Konfigurations-Metamodell ermöglicht zunächst die Konfiguration, d.h. die Zusammenstellung der Dienstleistungskomponenten. Für die Konfiguration der nicht-funktionalen Eigenschaften [O'S06] können zwei Ansätze verfolgt werden. Einerseits kann für jede Ausprägung nicht-funktionaler Eigenschaften eine eigene Komponente spezifiziert werden. Dieser Ansatz entspricht der Weiterführung der Konfiguration der Komponenten. Nachteilig hieran ist allerdings, dass bereits eine relativ moderate Anzahl möglicher Ausprägungen die Notwendigkeit der Definition vieler Komponenten nach sich zieht. Die Anzahl notwendiger Komponenten ist das Produkt aus der Anzahl der Eigenschaften und der Gesamtzahl möglicher Ausprägungen der jeweiligen Eigenschaften.

Es erscheint daher sinnvoller, nicht-funktionale Eigenschaften sowie deren mögliche Ausprägungen ebenfalls als Knoten im Graphen darzustellen. Diese Knoten können dann Komponenten zugewiesen werden und ermöglichen eine einfache Aussage darüber, welche möglichen Eigenschaften und Ausprägungen diese besitzen. Dazu wird die Knotenmenge des Graphen um die beiden Typen *Characteristics* zur Definition von Attributen und *CharacteristicValues* zur Definition der Ausprägungen erweitert.

$$V = Components \cup Connectors \cup Characteristics \cup CharacteristicValues$$

Durch die Integration der beiden neuen Knotentypen ergeben sich ferner weitere Restriktionen an den Graphen. Zuerst dürfen Attributknoten nur als direkte Kinder von Komponenten verwendet werden.

$$\forall (v_1, v_2) \in E: v_2 \in Characteristics \Rightarrow v_1 \in Components$$

Weiterhin dürfen Attribute nur mittels eines Konnektors mit ihren möglichen Ausprägungen verbunden werden.

$$\forall (v_1, v_2) \in E: v_1 \in Characteristics \Rightarrow$$
$$v_2 \in Connectors \wedge postnodes(v_2) \subseteq CharacteristicValues$$

Ausprägungen selber dürfen als direkte Elternelemente nur Konnektoren besitzen, die wiederum nur mit Attributen verbunden sein dürfen. Dazu wird das Hilfskonstrukt *prenodes* eingeführt. Dieses ermittelt analog zur Funktion *postnodes* die Vorgängerknoten eines Elements.

$$prenodes: V \rightarrow \wp(V)$$
$$prenodes(v) = \{v' \in V: \exists e \in E: e = (v', v)\}$$
$$\forall (v_1, v_2) \in E: v_2 \in CharacteristicValues \Rightarrow$$
$$v_1 \in Connectors \wedge prenodes(v_1) \subseteq Characteristics$$

Schließlich ist noch festzulegen, dass Ausprägungsknoten keine weiteren Kindelemente besitzen dürfen.

$$\forall (v_1, v_2) \in E: v_1 \notin CharacteristicValues$$

erschienen im Tagungsband der INFORMATIK 2011
Lecture Notes in Informatics, Band P192
ISBN 978-3-88579-286-4
weitere Artikel online:
http://informatik2011.de/519.html

3 Fazit

In diesem Artikel wurde der Ansatz der Modularisierung von Dienstleistungen formal vorgestellt. Das dadurch entstandene Metamodell zur Modellierung von Dienstleistungskomponenten sowie deren Abhängigkeiten untereinander dient als Grundlage zur Konfiguration von Dienstleistungen sowie als Ausgangspunkt weiterer Überlegungen. Um die Übertragbarkeit der vorgestellten Ansätze in die Praxis zu demonstrieren wurde ein Softwarewerkzeug entwickelt, welches das Metamodell implementiert. Dieses wird iterativ ausgebaut und soll am Ende als zentrales Werkzeug zur Modellierung und Konfiguration von Dienstleistungen dienen. Dieses ermöglicht es Mitarbeitern aus Fachabteilungen, Dienstleistungskomponenten auf einfache Art und Weise zu spezifizieren. Bereits durch diese Nutzung des Komponentenkonzepts lässt sich die Komplexität der Dienstleistungsentwicklung verringern. Einerseits ermöglichen Komponenten eine genauere Betrachtung relevanter Eigenschaften. Andererseits lassen sich durch die formale Spezifikation redundante Prozesse aufspüren. Die praktische Anwendbarkeit des Werkzeugs wurde in der Vergangenheit bereits durch die Integration des Dienstleistungsportfolios einer großen Full-Service-Internet-Agentur geprüft.

Neben der reinen Definition ist vor allem die Konfiguration von Dienstleistungen basierend auf kundenindividuellen Anforderungen von Bedeutung. Durch Nutzung des Komponentenkonzepts kann in enger Kooperation und Interaktion mit den Kunden ein passendes Angebot erstellt werden. Um dies zu ermöglichen, müssen im Laufe der weiteren Untersuchungen Möglichkeiten identifiziert werden, die eine einfache Konfiguration ermöglichen. Hierbei lassen sich mehrere Arten unterscheiden. Auf der einen Seite kann die Konfiguration Top-Down bzw. Bottom-Up erfolgen. Im ersten Fall können Kunden ausgehend von vordefinierten Dienstleistungspaketen die spezifische Erbringung anpassen. Der zweite Fall versetzt Kunden in die Lage, sich aus einer Menge von Komponenten ein gänzlich individuelles Angebot zu erstellen. Andererseits ist es aber auch möglich, dialoggetriebene Konfigurationsszenarien zu entwickeln, wie sie im Bereich der Automobilindustrie bereits verbreitet sind.

Literaturverzeichnis

[AP06] Aalst, M. W. P. V. D.; Pesic, M.: Specifying, Discovering, and Monitoring Service Flows: Making Web Services Process-Aware. BPM Center Report. BPM Center, 2006

[AS10] Araujo, L.; Spring, M.: Complex performance, process modularity and the spatial configuration of production. In: (Caldwell, N.; Howard, M. Hrsg.) Procuring Complex Performance: Studies in Innovation in Product-Service Management. London: Routledge, 2010.

[BC97] Baldwin, C. Y.; Cklark, K. B.: Managing in an Age of Modularity. Harvard Business Review, 1997, 75, S. 84-93.

[Be09] Becker, J.; Beverungen, D.; Knackstedt, R.; Müller, O.: Konzeption einer Modellierungssprache zur softwarewerkzeugunterstützten Modellierung, Konfiguration und Bewertung hybrider Leistungsbündel. In: (Thomas, O.; Nüttgens, M. Hrsg.) Dienstleistungsmodellierung - Methoden, Werkzeuge und Branchenlösungen. Berlin, Heidelberg: Physica-Verlag, 2009.

[BO03] Becker, S.; Overhage, S.: Stücklistenbasiertes Komponenten-Konfigurationsmanagement. In: (Turowski, K. Hrsg.) Tagungsband des 5. Workshops Komponentenorientierte betriebliche Anwendungssysteme (WKBA 5). Augsburg, 2003.

[Bö04] Böhmann, T.: Modularisierung von IT-Dienstleistungen : eine Methode für das Service-Engineering, Wiesbaden, Deutscher Universitätsverlag, 2004.

[BK11] Böttcher, M.; Klingner, S.: Der Komponentenbegriff der Dienstleistungsdomäne. In: (Fähnrich, K.-P.; Franczyk, B. Hrsg.) Service Science - Neue Perspektiven für die Informatik, 27.09. -01.10.2010 2010 Leipzig. Gesellschaft für Informatik (GI), 2011, S. 59-66.

[BSF11] Böttcher, M.; Swialkowski, R.; Fähnrich, K.-P.: Produktivitätsbetrachtung bei der Komponentisierung von Dienstleistungen. In: (Gatermann, I.; Fleck, M. Hrsg.) Mit Dienstleistungen die Zukunft gestalten – Impulse aus Forschung und Praxis. Campus Verlag, 2011.

[Bu04] Budinsky, F.; Steinberg, D.; Merks, E.; Ellersick, R.; Grose, T. J.: Eclipse Modeling Framework, Boston et al., Addison-Wesley, 2004.

[Bu07] Burianek, F.; Ihl, C.; Bonnemeier, S.; Reichenwald, R.: Typologisierung hybrider Produkte - Ein Ansatz basierend auf der Komplexität der Leistungserbringung. In: (Reichenwald, R. Hrsg.) Arbeitsberichte des Lehrstuhls für Betriebswirtschaftslehre - Information, Organisation und Management. München: Technische Universität München, 2007.

[Bu02] Burr, W.: Service Engineering bei technischen Dienstleistungen, Wiesbaden, Deutscher Universitäts-Verlag, 2002.

[CDG06] Corsten, H.; Dresch, K.-M.; Gössinger, R.: Modularisierung von Dienstleistungen - Eine koordinationsorientierte Analyse. In: (Corsten, H. Hrsg.) Schriften zum Produktionsmanagement. Kaiserslautern: Lehrstuhl für Produktionswirtschaft, Universität Kaiserslautern, 2006.

[CE00] Czarnecki, K.; Eisenecker, U.: *Generative Programming: Methods, Tools and Application*, Addison-Wesley Professional, 2000.

[DBF01] Da Silveira, G.; Borenstein, D.; Fogliatto, F. S.: Mass customization: Literature review and research directions. International Journal of Production Economics, 2001, 72**,** S. 1-13.

[DO00] Dierks, H.; Olderog, E.-R.: Temporale Spezifikationslogiken. at - Automatisierungstechnik, 2000, 48**,** 7-10.

[Do03] DOWNAR, P.: Vermarktung industrieller Dienstleistungen, Wiesbaden, Deutscher Universitätsverlag, 2003.

[Em05] Emmrich, A.: Ein Beitrag zur systematischen Entwicklung produktorientierter Dienstleistungen. Dr. rer. pol. Promotion, Universität Paderborn, 2005.

[Fa05a] Favre, J.-M.: Foundations of Meta-Pyramids: Languages vs. Metamodels -- Episode II: Story of Thotus the Baboon1. In: (Bezivin, J.; Heckel, R. Hrsg.) Language Engineering for Model-Driven Software Development. Dragstuhl: Internationales Bildungs- und Forschungszentrum für Informatik (IBFI), 2005.

[Fa05b] Favre, J.-M.: Foundations of Model (Driven) (Reverse) Engineering: Models -- Episode I: Stories of The Fidus Papyrus and of The Solarus. In: (Bezivin, J.; Heckel, R. Hrsg.) Language Engineering for Model-Driven Software Development. Dragstuhl: Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), 2005.

[FL97] Feitzinger, E.; Lee, H. L.: Mass Customization at Hewlett-Packard: The Power of Postponement. Harvard Business Review, 1997, January-February**,** 116-121.

[Ha05] Harold, E. R.: XML in a Nutshell, Bejing, Köln, et al., O'Reilly, 2005.

[Hi08] Hitzler, P.: Semantic Web: Grundlagen, Berlin, Springer, 2008.

[Ka90] Kang, K., Cohen, S., Hess, J., Nowak, W.; Peterson, S.: Feature-oriented domain analysis (FODA) feasibility study. Pittsburgh: Carnegie Mellon University, 1990.

[Ke09] Kern, H., Böttcher, M., Kühne, S.; Meyer, K.: Ansatz zur ganzheitlichen Erzeugung und Verarbeitung von Dienstleistungsmodellen. In: (Thomas, O.; Nüttgens, M. Hrsg.) Dienstleistungsmodellierung - Methoden, Werkzeuge und Branchenlösungen. Berlin, Heidelberg: Physica-Verlag, 2009.

[Kr95] Krajicek, J.: Bounded Arithmetic, Propositional Logic, and Complex Theory, Cambridge, Cambridge University Press, 1995.

[Kü04] Kühne, T.: What is a Model? In: Language Engineering for Model-Driven Software Development, 2004 Schloss Dagstuhl, Germany.

[Lo04] Lorenz-Meyer, D.: Management industrieller Dienstleistungen, Wiesbaden, Deutscher Universtitäts-Verlag, 2004.

[O'S06] O'Sullivan, J.: Towards a Precise Understanding of Service Properties. Doctor of Philosophy PhD, Queensland University of Technology, 2006.

[PS00] Peters, L.; Sadin, H.: IT and the mass customiziation of services: the challenges of implementation. International Journal of Information Management, 2000, 20**,** S. 103-119.

[Pi99] Pine II, J. B.: Mass Customization: The Frontier in Business Competition, Boston, MA, Harvard Business School Press, 1999.

[Sa97] Sametinger, J.: Software Engineering with Reusable Components, Berlin, Heidelberg, et al., Springer, 1997.

[Sc02] Scheer, A.-W.: ARIS - Vom Geschäftsprozess zum Anwendungssystem, Berlin et al., Springer, 2002.

[St65] Starr, M. K.: Modular Production - A New Concept. Harvard Business Review, 1965, 43**,** S. 131-142.

[SB02] Svensson, C.; Barfod, A.: Limits and opportunities in mass customization for "build to order" SMEs. Computers in Industry, 2002, 49**,** S. 77-89.

[Sz02] Szyperski, C.: Component software - beyond object-oriented programming, London et al., Addison-Wesley, 2002.

[Va62] Vazsonyi, A.: Die Planungsrechnung in Wirtschaft und Industrie, Wien et al., Oldenbourg, 1962.

erschienen im Tagungsband der INFORMATIK 2011
Lecture Notes in Informatics, Band P192
ISBN 978-3-88579-286-4
weitere Artikel online:
http://informatik2011.de/519.html