

Flexibilitätsanalyse service-orientierter Architekturen zur Realisierung von Geschäftsprozessen

Guido Laures

Hasso-Plattner-Institut für Softwaresystemtechnik GmbH
Prof.-Dr.-Helmert-Strasse 2-3, 14482 Potsdam, Germany
guido.laures@hpi.uni-potsdam.de

Abstract: Die Effizienz eines Geschäftsprozesses hängt zum großen Teil von dessen Automatisierungsgrad ab. Aus diesem Grund gehört Prozessautomation zu einer der wichtigsten Erfolgsfaktoren innerhalb einer IT-Organisation. Durch die Einführung einer unternehmensweiten, service-orientierten Architektur versprechen sich Unternehmen eine höhere Flexibilität.

Beim Design und der Implementierung eines Service können jedoch Fehler begangen werden, die dazu führen, dass die Investitionen in die Einführung von Services nicht in gesteigerter Prozesseffizienz münden und damit das ursprüngliche Ziel verfehlen. In dieser Arbeit werden Muster innerhalb einer service-orientierten Architektur auf ihre Flexibilität hin untersucht und bewertet. Die dabei entstehende Muster-sammlung unterstützt IT-Architekten beim Service- und Prozessdesign und hilft damit a priori unflexible IT-Architekturen zu vermeiden.

1 Einführung

Die Optimierung der Flexibilität der IT-seitigen Realisierung von Geschäftsprozessen ist oft eines der Hauptziele bei der Einführung einer unternehmensweiten service-orientierten Architektur (SOA) [PvdH06]. Die Vorstellungen davon, was in diesem Zusammenhang genau unter Flexibilität zu verstehen ist, divergieren jedoch signifikant. Was fehlt, ist eine Kategorisierung der architekturellen Konzepte, die innerhalb einer SOA üblicherweise verwendet werden, und deren Bewertung nach formalen Flexibilitätskriterien. Die daraus ableitbaren Erkenntnisse können zukünftig vermeiden, dass schon in frühen Phasen eines SOA-Projektes Fehler begangen werden, die die angestrebten Projektziele gefährden.

Diese Arbeit lenkt den Blick beim Entwurf und der Realisierung einer Unternehmens-SOA auf die Flexibilität der Implementierung der umzusetzenden Geschäftsprozesse. Hierzu wird nach Abgrenzung des Begriffs der Unternehmensarchitektur eine Definition sowie eine Berechnungsvorschrift für Flexibilität gegeben. Darauf beziehend werden Architekturmuster erarbeitet, auf Flexibilität hin untersucht und anhand einer Fallstudie erläutert. Die erarbeiteten Muster werden daraufhin zusammenfassend und vergleichend gegenübergestellt und bewertet bevor themenverwandte Arbeiten gegen den in diesem Papier vorgestellten Ansatz diskutiert werden. Schließlich wird ein Ausblick gegeben, wie die in dieser Arbeit vorgestellten Erkenntnisse weitergeführt und ausgeweitet werden können.

2 Grundlagen

2.1 Unternehmensarchitekturen

Unternehmensarchitekturen werden von Zachmann [Zac97] als das *Thema des Jahrhunderts* bezeichnet. Der Begriff *Architektur* wird hier als die Menge der Entwurfsartefakte und beschreibender Repräsentationen bezeichnet, die notwendig sind, um ein Objekt so zu beschreiben, dass es mit gleicher Qualität reproduziert und über seine Lebensdauer gewartet werden kann. Ein Unternehmen (Enterprise) ist nach Wegmann [Weg03] eine Organisation von Ressourcen, die Prozessen unterliegt. Beispiele für solche Ressourcen sind Menschen, Computer, Maschinen oder auch Gebäude. Eine Unternehmensarchitektur (Enterprise Architecture) ist eine reproduzierbare und wartbare Beschreibung eben solcher Ressourcen.

Da Unternehmensarchitekturen der Untersuchungsgegenstand dieser Arbeit sind, ist es wichtig, sie von Architekturen zur Implementierung von Unternehmensanwendungen abzugrenzen. Die ebenfalls oft als Enterprise Frameworks bezeichneten Technologien wie J2EE oder .NET bilden nach unserem Verständnis keine Unternehmensarchitekturen. Gleiches gilt für Architekturen von Unternehmensanwendungen wie sie beispielsweise Fowler [Fow02] behandelt werden. Diese beschreiben applikationsinterne Architekturkonzepte, die im Gegensatz zu applikationsübergreifenden für eine Unternehmensarchitektur nicht relevant sind. Der folgende Abriss der historischen Entwicklung von Unternehmensarchitekturen soll dieses Verständnis weiter untermauern.

Service-orientierten Architekturen sollen in IT-Projekten den Fokus weg von technischen hin zu prozessrelevanten Aspekten der Unternehmensarchitektur leiten. Technische Aspekte bestimmen jedoch in SOA Projekten oft den Planungs- und Projektalltag und priorisieren technische Integration höher als die Transparenz von Unternehmensprozessen in den IT-Systemen der Unternehmensarchitektur. Auch bei der Verwendung von Services zur Abbildung von Geschäftsfunktionalitäten besteht die Gefahr, unflexible Gesamtarchitekturen zu erhalten. Es geht nicht nur um die korrekte, technologische Bereitstellung von Services, sondern auch um deren strukturierte Einbindung und Verwendung innerhalb der Unternehmensarchitektur.

2.2 Flexibilität

Die Abbildung von Geschäftsprozessen in den Strukturen der sie realisierenden IT-Landschaften ist sowohl modellierungsseitig als auch in Bezug auf Wartung eine der größten Herausforderungen der IT-Organisation eines Unternehmens [DS90, DHL01, LRS01]. Formale Kriterien zur Bewertung der Güte einer Reflektion von Geschäftsprozessen innerhalb einer IT-Architektur sind jedoch schwer zu finden. Das Ziel der Flexibilisierung wird zwar allgemein hin als erstrebenswert hingestellt [PvdH06, NJFM96, Zac99] jedoch wird dessen Erreichungsgrad selten formal geprüft.

In [Wik06] wird Flexibilität als die Fähigkeit bezeichnet, sich an verschiedene Begebenheiten anzupassen. Auf Unternehmensarchitekturen und Unternehmensprozesse bezogen



Abbildung 1: Die Beispielprozesse p_1 und p_2 bestehen lediglich aus einer simplen Sequenz von zwei Tasks

bedeutet Flexibilität also die Fähigkeit einer IT-Landschaft, auf ändernde Geschäftsprozesse hin angepasst zu werden.

Die Flexibilität eines Systems bzgl. der Umsetzung von neuen oder modifizierten Geschäftsprozessen verhält sich anti-proportional zu den Aufwänden, die für diese Umsetzung betrieben werden müssen. Zudem sollte zur Bestimmung der Flexibilität auch die resultierende Wartbarkeit eines Systems nach Umsetzung notwendiger Modifikationen berücksichtigt werden. Hierzu gilt die Regel, dass die Flexibilität als schlechter zu bewerten ist, sofern die Wartbarkeit des Systems nach Umsetzung einer Änderung in der Prozesslandschaft gesunken ist.

Sei S_P eine Menge von Systemen, die die Prozessmenge P realisiert. Sei P' eine weitere Prozessmenge und $T_{S_P, S_{P'}}$ die Menge der Transformationsschritte, die S_P durchlaufen muss, um es in $S_{P'}$ zu überführen. Seien weiterhin $E_{T_{S_P, S_{P'}}}$ die Summe der Aufwände der Elemente von $T_{S_P, S_{P'}}$ und M_S die Wartbarkeit einer Systemmenge S . Dann bezeichnet

$$F_{S_P, P'} = \frac{M_{S_{P'}}}{1 + E_{T_{S_P, S_{P'}}}}$$

die Flexibilität von S_P bzgl. einer Überführung in $S_{P'}$.

Hierbei wird davon ausgegangen, dass $E_{T_{S_P, S_{P'}}} \geq 0$ gilt und damit

$$T_{S_P, S_{P'}} = \{\} \Rightarrow E_{T_{S_P, S_{P'}}} = 0 \Rightarrow F_{S_P, P'} = M_{S_{P'}}$$

sicherstellt. Zudem wird festgelegt, dass $0 < M \leq 1$ und damit auch $0 < F \leq 1$ gilt.

2.3 Fallstudie

Das folgende Fallbeispiel wird im nächsten Abschnitt dazu verwendet, die unterschiedlichen Flexibilitätsstufen, die innerhalb service-orientierter Architekturen herrschen können, zu diskutieren. Das Beispiel setzt voraus, dass Tasks innerhalb von Geschäftsprozessen grundsätzlich auf Services innerhalb einer Unternehmens-SOA abgebildet werden können. Diese Voraussetzung ist zwar eher unrealistisch, da sich derzeit die meisten Unternehmen erst auf dem Weg hin zu einer solchen Architektur befinden [STM⁺04]. Es zeigt sich jedoch anschaulich, dass selbst in Unternehmen, in denen diese Voraussetzung gilt, die verwendeten Muster zur Service-Realisierung über den Flexibilitätsgrad entscheidet [Hef04].

Im untersuchten Unternehmen sei die Prozessmenge P bestehend aus zwei sehr einfachen Geschäftsprozessen p_1 und p_2 gegeben (siehe Abbildung 1). Prozesse sind in dieser Arbeit



Abbildung 2: Der neu eingeführte Prozess p_3 komponiert Tasks der Prozesse p_1 und p_2

in BPMN [OMG06] spezifiziert. In Zukunft sollen die Systeme des Unternehmens jedoch die Prozessgruppe $P' = P \cup \{p_3\}$ unterstützt werden. Der neu eingeführte Prozess p_3 beinhaltet Tasks aus beiden Prozessen aus P . Dies ist ein übliches Szenario, bei dem existierende Tasks mit neuen zu einem neuen Prozess komponiert werden. Der neu eingeführte Prozess ist in Abbildung 2 dargestellt.

Im folgenden werden Realisierungsmuster der Prozessgruppe P daraufhin untersucht, wie flexibel sie in Bezug auf eine Realisierung der Prozessgruppe P' sind.

2.4 Notation

Zur grafischen Darstellung der systemseitigen Realisierung eines Prozesses wird die Notation aus Tabelle 1 verwendet. Das dabei verwendete Konzept für systemseitige Serviceimplementierungen ist angelehnt an den Agentenbegriff aus der Web Service Architecture der W3C [W3C04].

3 Muster

In diesem Abschnitt werden eine Reihe von Mustern vorgestellt, die innerhalb einer Unternehmensarchitektur vorliegen können. Wir beschränken uns bei der Identifikation von Mustern jedoch nicht auf die technologische Struktur der Systeme und deren Abhängigkeiten. Vielmehr stellen wir diese in Relation zu den Geschäftsprozessen, die durch sie implizit oder explizit realisiert werden. Nach einer allgemeinen Beschreibung jedes Musters, wenden wir es auf die Fallstudie an, um es in den Kontext von SOA zu stellen. Abschließend wird die Flexibilität jedes Musters untersucht und anhand der oben gegebenen Formel errechnet.

3.1 Direkter Aufruf

Das erste untersuchte Muster ist wohl aufgrund seiner Trivialität auch das am häufigsten in einer Unternehmensarchitektur auftretende. Beim Muster *Direkter Aufruf* wird eine Transition zwischen zwei Tasks t_1 und t_2 innerhalb eines Geschäftsprozess direkt vom realisierenden System a_1 des Tasks t_1 implementiert (vgl. Abbildung 3). Das System a_1 ruft also das System a_2 direkt auf, um somit implizit den Prozess zu implementieren.







| | | |
|---|---------------------------------------|---|
|  | System | Innerhalb einer Unternehmens-SOA ist dies meist eine Geschäftsanwendung. |
|  | Service | System, das den Service-Agenten implementiert. Für gewöhnlich ist dies nicht gleichzusetzen mit dem System, das die eigentliche Service-Funktionalität implementiert. |
|  | Service-Realisierung | Service-Aufrufe von s_1 werden von a_1 realisiert. a_1 implementiert also die Geschäftsfunktionalität, die zur Realisierung der von s_1 bereitgestellten Service-Schnittstelle benötigt wird. |
|  | Trigger | Nicht näher spezifizierter interner oder externer Trigger. Dies kann beispielsweise ein anderes System oder auch ein interner Zeitgeber sein. |
| --- | semantischer Bezug | Semantischer Bezug von Prozesselementen und deren systemseitiger Realisierung. |
|  | systemseitiger Prozessfluss | Systemseitiger Fluss des realisierten Geschäftsprozesses. Die Nummer bestimmt die Reihenfolge innerhalb des Fluss. |
|  | systemseitige Prozessfluss-Beendigung | Symbolisiert die Stelle, an der ein Prozessfluss durch die Rücklieferung eines Ergebnis oder durch die Ausführung einer Aktion beendet wird. |

Tabelle 1: Verwendete Notation

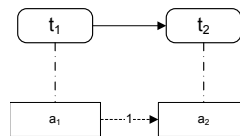


Abbildung 3: Direkter Aufruf

3.1.1 Fallstudie

Wendet man dieses Muster auf obige Fallstudie an, so ergibt sich für die Prozessgruppe P innerhalb einer service-orientierten Architektur die Realisierungen der Abbildung 4. Da wir in unserer Fallstudie von einer service-orientierten Architektur ausgehen, sind die aufgerufenen Systeme jetzt die zu den Tasks des Geschäftsprozesses semantisch äquivalenten Services. Würde im Prozess p_1 a_1 nicht s_2 sondern stattdessen direkt a_2 aufrufen, so könnte man nicht mehr von Service-Orientierung sprechen.

Es ist an dieser Stelle erwähnenswert, dass a_1 zur Realisierung der dargestellten Prozesse zur Laufzeit über Prozesskontextinformationen, also Informationen darüber, welcher Prozess gerade durchlaufen wird, verfügen muss. Dies ist deshalb der Fall, da eine Laufzeitentscheidung getroffen werden muss, ob bei Aufruf des Service s_1 der Service s_2 oder

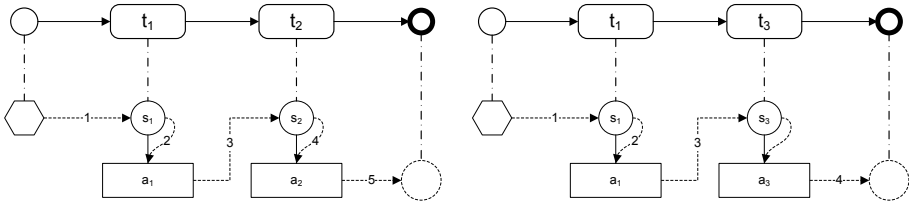


Abbildung 4: *Direkter Aufruf* innerhalb einer SOA

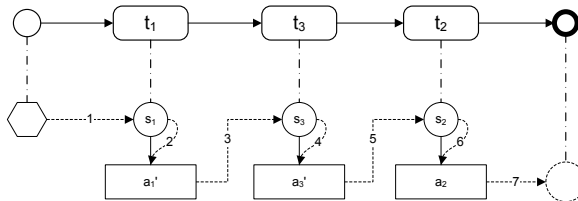


Abbildung 5: Realisierung von p_3 mit Muster *Direkter Aufruf*

der Service s_3 aufgerufen werden muss. Diese Information muss beim Service-Aufruf von s_1 mitgegeben werden. Eine solche implizite Abhängigkeit von Prozesskontextinformationen wirkt sich offensichtlich negativ auf die Wartbarkeit eines Systems aus.

3.1.2 Flexibilitätsbewertung

Um die Flexibilität dieses Musters zu bewerten, betrachten wir nun das Verhalten der systemseitigen Realisierung bei Einführung des neuen Prozesses p_3 . Abbildung 5 zeigt, dass a_3 modifiziert werden muss, um den neuen Prozess realisieren zu können. Dies ist dadurch begründet, dass a_3 bisher nach Aufruf durch s_3 den Prozessfluss beendete (p_2). In p_3 jedoch muss a_3 s_2 aufrufen, um den Prozess korrekt abzubilden. Dies wird wieder durch Kontextinformationen im Aufruf von s_3 realisiert. Unter Umständen muss auch a_1 modifiziert werden. Dies wird dann notwendig, wenn die von a_3 benötigten Kontextinformationen nicht mittels der bestehenden Implementierung von a_1 durchgeleitet werden können.

Um den Wert der Flexibilität dieses Musters zu bestimmen und mit den nachfolgenden Mustern zu vergleichen werden nun für bestimmte Größen Referenzwerte festgelegt. Die Wartbarkeit des Musters *Direkter Aufruf* (DA) bezeichnen wir mit M_{DA} , den Aufwand zur Überführung von S_P nach $S_{P'}$ als E_{DA} und erhalten gemäß obiger Formel als Wert für die Flexibilität dieses Musters

$$F_{DA} = \frac{M_{DA}}{1 + E_{DA}}.$$

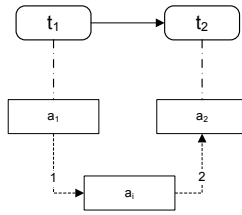


Abbildung 6: Indirekter Aufruf

3.2 Indirekter Aufruf

Das Muster *Indirekter Aufruf* fügt dem Muster *Direkter Aufruf* eine Indirektion hinzu. Zwischen dem System a_1 , das den Task t_1 implementiert, und a_2 , das den darauffolgenden Task t_2 implementiert liegt hier ein zusätzliches System a_i , das von a_1 aufgerufen wird und den Aufruf nach a_2 weiterleitet (vgl. Abbildung 6). Das weiterleitende System kann demnach auch als Proxy [GHJV94] bezeichnet werden.

3.2.1 Fallstudie

Es kommt oft vor, dass a_i Datentransformationen durchführt, um Inkompatibilitäten von Datenformaten zu überbrücken. Es ist somit ein typisches Szenario, wie man es in Architekturen antrifft, die EAI Produkte verwenden. Die Funktionalität von a_i ist eine typische Eigenschaft eines solchen EAI Produktes wie beispielsweise eines Integration Broker. Abbildung 7 veranschaulicht die Realisierung von P' mittels des Musters *Indirekter Aufruf*. Auch hier wird davon ausgegangen, dass a_i die vorliegende SOA aus der Fallstudie verwendet, um indirekte Aufrufe zu realisieren. Dies bietet sich schon deshalb an, da die meisten EAI Produkte bereits in der Lage sind, gängige Service-Schnittstellen und Protokolle (z.B. SOAP) zu bedienen. Wenn a_i direkt auf die Geschäftsanwendungen a_2 und a_3 verweisen würde, würde man diese Eigenschaft ignorieren und Mehraufwand bei der Realisierung von Integrationsszenarien in Kauf nehmen.

3.2.2 Flexibilitätswertung

Durch die Explizitmachung der Kommunikation zwischen den Geschäftsanwendungen innerhalb von a_i wird auch die Realisierung der Prozesstransformationen sichtbar und damit leichter zu warten. Jedoch wird die Systemlandschaft durch die Hinzunahme des zusätzlichen Systems a_i komplexer und deren Wartung erfordert zusätzliches Produktwissen. Somit kann man die Wartbarkeit M_{IA} dieses Musters mit der des Musters *Direkter Aufruf* M_{DA} gleichsetzen ($M_{DA} = M_{IA}$). Es ist zudem davon auszugehen, dass a_i darauf ausgelegt ist, exakt solche Änderungen zu unterstützen, die notwendig sind, um P' zu unterstützen, da dies eine Kernfunktionalität von Integration Brokern darstellt. Der Aufwand E_{IA} zur Realisierung von P' in diesem Muster ist somit geringer als E_{DA} . Hiermit ergibt

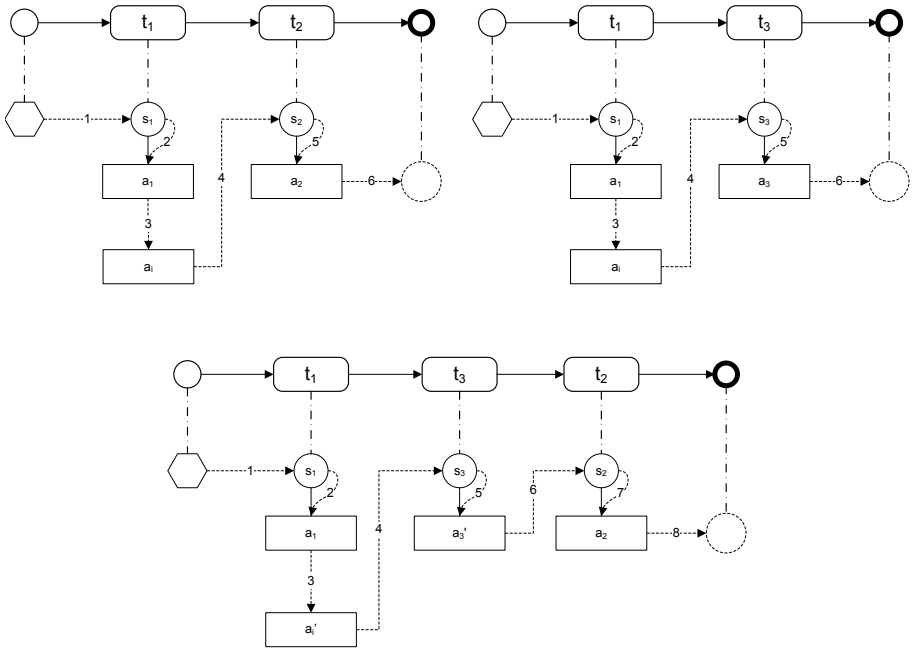


Abbildung 7: Realisierung von P' mit Muster *Indirekter Aufruf*

sich für die Flexibilität F_{IA} dieses Musters

$$F_{IA} - F_{DA} = \frac{M_{DA}}{1 - E_{IA}} - \frac{M_{DA}}{1 - E_{DA}} > 0,$$

da $E_{IA} < E_{DA}$, womit gezeigt ist, dass dieses Muster eine höhere Flexibilität bietet als das Muster *Direkter Aufruf*.

3.3 Vermittelter Aufruf

Beim Muster *Vermittelter Aufruf* konsultiert das System a_1 ein weiteres System a_{ref} bevor es selbst den Aufruf von a_2 durchführt (vgl. Abbildung 8). Hierbei wird vorausgesetzt, dass die Informationen, die a_{ref} beim Aufruf durch a_1 liefert, erst den Endpunkt des daraufhin durchgeführten Aufrufs von a_2 durch a_1 festlegt (Grounding). Dadurch ist es möglich, den Aufruf von a_1 auf ein anderes System umzuleiten, ohne dafür Anpassungen an a_1 vornehmen zu müssen. Dies setzt jedoch voraus, dass die von a_{ref} zurückgelieferte Referenz schnittstellenkompatibel zu a_2 ist.

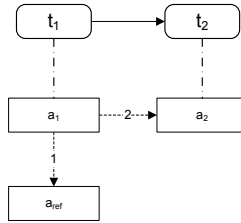


Abbildung 8: Vermittelter Aufruf

3.3.1 Fallstudie

Die Realisierung von p_3 ist in Abbildung 9 dargestellt. Das vermittelnde System ist im Falle einer SOA das Service-Verzeichnis. Erst in diesem Muster wird eine SOA (inkl. Verzeichnis) gemäß der ursprünglichen Konzeption aus [IBM00] erreicht.

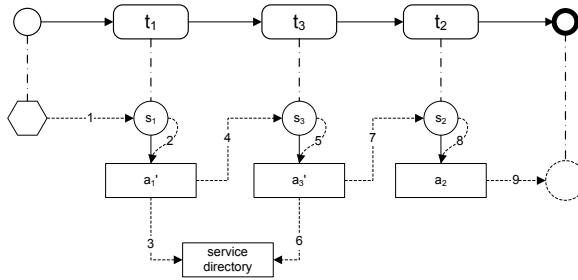


Abbildung 9: Realisierung von p_3 mit Muster *Vermittelter Aufruf*

3.3.2 Flexibilitätsbewertung

Die Wartbarkeit M_{VA} der resultierenden Architektur bei Anwendung des Musters *Vermittelter Aufruf* ist vergleichbar mit der der ersten beiden Muster ($M_{VA} = M_{DA} = M_{IA}$). Auch hier wird im Vergleich zum Muster *Direkter Aufruf* ein zusätzliches System (a_{ref}) benötigt (verringerte Wartbarkeit), womit jedoch gleichzeitig eine Explizitmachung der Prozesstransitionen von t_1 erreicht werden kann, die diesen Wartbarkeitsnachteil aufwiegt.

Der Aufwand E_{VA} zur Realisierung von P' ist in diesem Muster höher als beim Muster *Indirekter Aufruf*. Es müssen die gleichen Systeme angepasst werden wie beim Muster *Direkter Aufruf* womit $E_{VA} = E_{DA}$ gilt. Damit ergibt sich für die Flexibilität dieses Musters

$$F_{VA} = \frac{M_{VA}}{1 + E_{VA}} = \frac{M_{DA}}{1 + E_{DA}} = F_{DA}.$$

Eine höhere Flexibilität wird dann erreicht, wenn es mehrere Implementierungen ein und der selben Service-Schnittstelle gibt, die dem vermittelnden System bekannt sind. Da wir

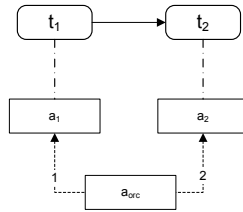


Abbildung 10: Zentraler Orchestrator

in unserem Fallbeispiel davon ausgehen, dass s_3 nicht kompatibel zu s_2 ist, folgen daraus auch bei diesem Muster die gleichen Bedingungen bzgl. Anpassungen zur Realisierung der Prozessgruppe P' wie beim Muster *Direkten Aufruf*.

Es mag an dieser Stelle überraschen, dass sich die Einführung eines Service Directory nicht positiv auf die Flexibilität auswirkt. Dies liegt daran, dass wir Flexibilität bezogen auf die Realisierung von Geschäftsprozessen untersuchen. In Bezug auf die technische Infrastruktur, bietet ein Service Directory selbstverständlich flexibilitätssteigernde Möglichkeiten, wie z.B. das Umziehen einer Funktionalität zu einer anderen technischen Komponente, ohne die Notwendigkeit zur Modifikation ihrer Klienten (lose Kopplung). Dies gilt jedoch nicht für Modifikationen an Geschäftsprozessen, da es sich dabei nicht um strukturelle sondern funktionale Entitäten handelt. Solche Modifikation wirken sich auch immer auf Klienten aus. Abhilfe könnten hier semantische Methoden schaffen, die wir aber in diesem Papier nicht näher betrachten werden.

3.4 Zentraler Orchestrator

Beim Muster *Zentraler Orchestrator* besteht keine direkte Abhängigkeit mehr zwischen den task-realisierenden Systemen. Statt dessen wird davon ausgegangen, dass die Implementierung eines Tasks atomar von Systemen zur Verfügung gestellt wird und damit keinerlei prozessrelevanten Seiteneffekte beim Aufruf eines solchen Systems auftreten. Die Steuerung des Prozesses wird von einem separaten System vorgenommen, dass die Aufrufe der Systeme gemäß der vorliegenden Geschäftsprozesse koordiniert (orchestriert) (vgl. Abbildung 10).

3.4.1 Fallstudie

Dieses Muster entspricht der architekturellen Idee einer ausgereiften, prozessbefähigten SOA [KBS04]. Abbildung 11 zeigt die Realisierung von P' aus dem Fallbeispiel. Bemerkenswert hier ist, dass in dieser Realisierung keine Prozesskontextinformationen in den Systemen a_1 und a_2 mehr notwendig sind. Dies ist darin begründet, dass die Aufrufe der Services, die von diesen Systemen bereitgestellt werden, atomar sind und keine impliziten Prozesse implementieren.

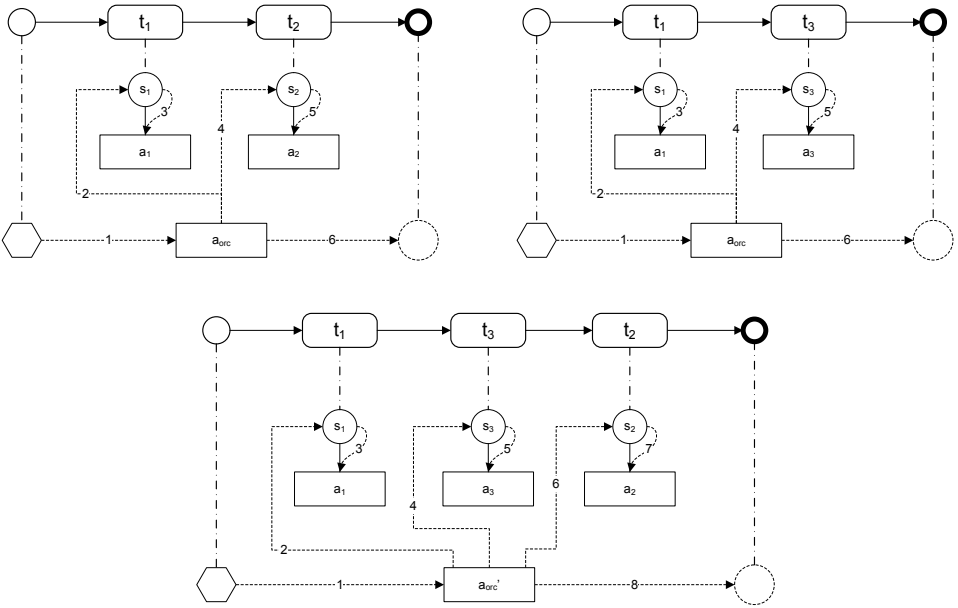


Abbildung 11: Realisierung von P' mit Muster *Zentraler Orchestrator*

3.4.2 Flexibilitätswertung

Durch dieses Realisierungsmuster ist die Wartbarkeit M_{ZO} der Systeme des Musters *Zentraler Orchestrator* signifikant höher als bei den vorher diskutierten Mustern ($M_{ZO} > M_{DA,IA,VA}$). Es ist nun möglich, Änderungen im Prozess zentral im System a_{orc} zu pflegen und die Systeme a_1 und a_3 unangetastet zu lassen. Die Realisierung des neuen Prozesses p_3 zieht lediglich Änderungen in a_{orc} mit sich. Innerhalb einer SOA wird a_{orc} sinnvollerweise ein System zur Pflege und zur Verwaltung von Geschäftsprozessen sein (Business Process Management, BPM), was zu einem signifikant geringerem Aufwand zur Umsetzung neuer Prozesse führt. Somit gilt für den Aufwand E_{ZO} zur Realisierung von P' mittels des Musters *Zentraler Orchestrator* $E_{ZO} < E_{IA}$ und folgerichtig der bisher beste Flexibilitätswert

$$F_{ZO} = \frac{M_{ZO}}{1 + E_{ZO}} > \frac{M_{IA}}{1 + E_{IA}} = F_{IA}.$$

3.4.3 Variante: Zentraler Orchestrator mit Vermittlung

Bei der Mustervariante *Zentraler Orchestrator mit Vermittlung* verwendet das orchestrierende System a_{orc} einen Vermittler a_{ref} zur Bestimmung des Groundings des Prozesses. Der Vorteil dieses Musters ist, dass die in a_{orc} verwendete Prozessbeschreibung zusätzlich von technischen, grounding-spezifischen Aspekten befreit werden kann, was den Prozess selbst wartbarer hält (vgl. Abbildung 12).

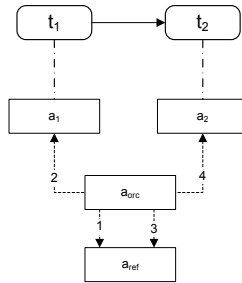


Abbildung 12: Zentraler Orchestrator mit Vermittlung

Diese Mustervariante ermöglicht ein Wechseln zu alternativen Groundings für bestimmte Tasks eines Prozesses, indem lediglich Grounding-Referenzen aus a_{ref} umgebogen werden. Die Flexibilität dieser Variante ist aufgrund der schlechteren Wartbarkeit durch Hinzunahme eines weiteren Systems etwas geringer als beim Muster *Zentraler Orchestrator*. Sie wird jedoch dann besser, wenn es für einen Service mehrere Implementierungen gibt, die je nach Prozess alternativ zueinander verwendet werden sollen.

4 Stand der Forschung

Die Reihe der Lösungsvorschläge zur Beherrschung wechselnder Geschäftsprozesse, die in bestehenden IT-Architekturen abgebildet werden müssen, reichen von Objekt-Orientierung [JEJ94] über Komponenten Frameworks [DW99] und persistenten Nachrichtensystemen [AMG95] bis hin zu den aktuellen Vorschlägen zur Verwendung von Web Services [LRS01] oder gar semantischen Web Service Architekturen [BFM02]. Auffällig hierbei ist, dass technologische Aspekte einen sehr hohen Stellenwert bei der Suche nach Lösungen einnehmen. Die Untersuchung von Prozessen und realisierenden Mustern nach formalen Aspekten ist jedoch kaum zu finden. Noch weniger findet man Belege für die Güte einer vorgeschlagenen Architektur in Form von formalen Berechnungen.

Eines der bedeutendsten Berechnungsmodelle zur Bestimmung der Flexibilität einer IT-Infrastruktur ist sicher das von Byrd und Turner [BT00]. Danach wird Flexibilität sogar als das entscheidende Maß für die Güte einer IT-Architektur herangezogen. Für unsere Betrachtungen war die Anwendung des von Byrd und Turner vorgeschlagenen Berechnungsmodells jedoch nicht geeignet, da sich dieses auf eine konkrete Instanz einer IT-Infrastruktur inklusive der Untersuchung von konkreten Werten einer spezifischen Unternehmensinstanz stützt. Wir benötigten für unsere Betrachtungen jedoch ein vereinfachtes Maß, das sich auch auf rein architekturelle Konzepte anwenden lässt. Zudem steht der Absolutheitsanspruch des aus diesem Berechnungsmodell resultierenden Flexibilitätswertes im Konflikt zu unserer Zielsetzung, lediglich die Flexibilität bezogen auf sich ändernde Geschäftsprozesse hin zu untersuchen. Dies gilt auch für das vom IEEE [IEE99] vorgeschlagene Flexibilitätmaß für Software.

Das von uns vorgeschlagene Berechnungsmodell lehnt sich an jene von Athey & Schmutzler [AS95] und auch Nelson & Ghods [NG98] an. Neben dem Wert für die Kosten (Aufwand) zur Realisierung einer Änderung, wurde bei unserer Berechnungsformel jedoch auch die aus der vorzunehmenden Änderung resultierende Wartbarkeit mit einbezogen.

Vergleichbar mit unserem Flexibilitätsmaß ist ebenfalls das aus dem Software-Engineering stammende Maß für die Kosten eines Evolutionsschritt von Software [EM05], aus dem die Flexibilität eines Software Systems abgeleitet werden kann. Dieses Maß haben wir auf die höhere Granularitätsstufe von Unternehmensarchitekturen gehoben und angepasst.

5 Schlussfolgerungen und Ausblick

Es wurde gezeigt, dass service-orientierte Architekturen zu einer höheren Flexibilität in Bezug auf die Realisierung einfacher Geschäftsprozesse führen können. Hierzu ist jedoch ein architektureller Entwurf gemäß des Musters *Zentraler Orchestrator* notwendig. Anhand der Muster *Direkter Aufruf*, *Indirekter Aufruf* und *Vermittelter Aufruf* wurde gezeigt, dass Services allein nicht hinreichend für die Erlangung einer erhöhten Flexibilität sind. Eine wichtige Erkenntnis der Untersuchung ist auch, dass die Einführung von Indirektionen keineswegs immer Vorteile in Bezug auf Flexibilität mit sich bringt. Abbildung 13 verdeutlicht fasst noch einmal die Werte der betrachteten Muster für die beiden flexibilitätsrelevanten Größen zusammen.

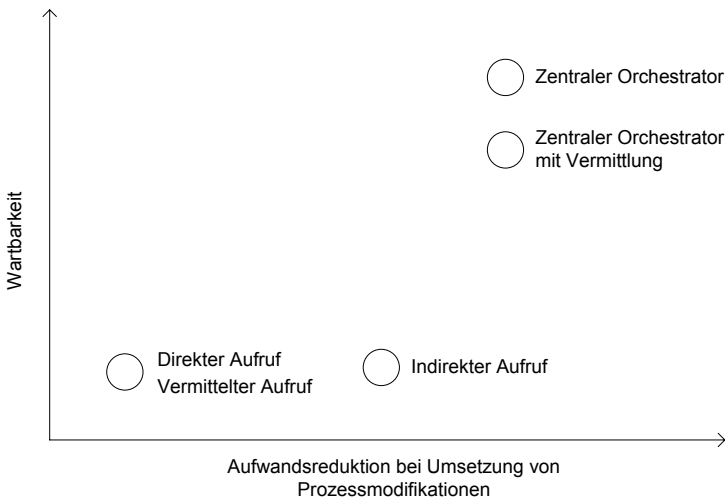


Abbildung 13: Zusammenfassung der Musterbewertung

Die von uns vorgestellten Muster erheben keinen Anspruch auf Vollständigkeit. Es sind sicherlich weitaus mehr Muster denkbar. So wäre sicherlich eine Flexibilitätsuntersuchung von Web Service Architekturen mit automatisierter Prozesskomposition unter Zuhilfenahme semantischer Methoden interessant.

Diese Arbeit legt die formale Grundlage für eine fundierte Bewertung der Flexibilität von Unternehmensarchitekturen zur Realisierung einfacher Prozesse, die lediglich aus Sequenzen besteht. Weitere Forschung ist notwendig, um eine vollständige Formalisierung der Flexibilitätswertung von Architekturen zur Realisierung beliebiger Prozesse durchführen zu können. In nachfolgenden Untersuchungen sollten demnach alle gängigen Muster von Workflows [vdAtHKB03], wie sie in Unternehmen zu finden sind, mit einbezogen werden.

Literatur

- [AMG95] G. Alonso, C. Mohan und R. Günthör. Exotica FMQM: A Persistent Message-Based Architecture for Distributed Workflow Management. In *Working Conference on Information Systems for Decentralized Organizations*, Trondheim, 1995.
- [AS95] Susan Athey und Armin Schmutzler. Product and Process Flexibility in an Innovative Environment. *RAND Journal of Economics*, 26(4):557–574, Winter, 1995.
- [BFM02] Christoph Bussler, Dieter Fensel und Alexander Maedche. A conceptual architecture for semantic web enabled web services. *SIGMOD Rec.*, 31(4):24–29, 2002.
- [BT00] Terry Anthony Byrd und Douglas E Turner. Measuring the Flexibility of Information Technology Infrastructure: Exploratory Analysis of a Construct. *Journal of Management Information Systems*, 17(1):167–208, 2000.
- [DHL01] Umeshwar Dayal, Meichun Hsu und Rivka Ladin. Business Process Coordination: State of the Art, Trends, and Open Issues. In *Proceedings of the 27th VLDB Conference*, Roma, Italy, 2001.
- [DS90] T. H. Davenport und J. E. Short. The new industrial engineering: Information technology and business process redesign. *Sloan Management review*, 31(4):11–27, 1990.
- [DW99] Desmond F. D’Souza und Alan Cameron Wills. *Objects, components, and frameworks with UML: the catalysis approach*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1999.
- [EM05] Amnon H. Eden und Tom Mens. Measuring Software Flexibility. *Technical report CSM-424*, 2005.
- [Fow02] Martin Fowler. *Patterns of Enterprise Application Architecture*. Addison-Wesley Professional, 2002.
- [GHJV94] Erich Gamma, Richard Helm, Ralph Johnson und John Vlissides. *Design Patterns*. Addison-Wesley, 1994.
- [Hef04] Randy Heffner. *Trends 2005: Service-Oriented Architecture And Web Services*. Forrester Research, Inc., 2004. <http://www.forrester.com>.
- [IBM00] IBM. *Web Services architecture overview*, 2000. <http://www-106.ibm.com/developerworks/webservices/library/w-ovr/>.
- [IEE99] IEEE. *Standard Glossary of Software Engineering Terminology 610.12-1990*, 1999.
- [JEJ94] I Jacobson, M Ericsson und A Jacobsons. *The object advantage: business process reengineering with object technology*. ACM Press/Addison-Wesley Publishing, 1994.

- [KBS04] Dirk Krafzig, Karl Banke und Dirk Slama. *Enterprise SOA*. Prentice Hall, 2004.
- [LRS01] F. Leymann, D. Roller, und M.-T. Schmidt. Web services and business process management. *IBM Systems Journal*, (41-2), 2001.
- [NG98] K. M. Nelson und M. Ghods. Measuring technology flexibility. *Eur. J. Inf. Syst.*, 7(4):232–240, 1998.
- [NJFM96] T. J. Norman, N. R. Jennings, P. Faratin und E. H. Mamdani. Designing and implementing a multi-agent architecture for business process management. *Intelligent Agents III, Springer*, Seiten 261–275, 1996.
- [OMG06] OMG. *Business Process Modeling Notation (BPMN) Specification*, 2006. <http://www.bpmn.org/>.
- [PvdH06] Michael P. Papazoglou und Willem-Jan van den Heuvel. Service-Oriented Design and Development Methodology. *International Journal of Web Engineering and Technology (IJWET)*, 2006.
- [STM⁺04] R. Schulte, J. Thompson, P. Malinverno, B. Lheureux und F. Kenney. *Predicts 2005: Application Integration, ESBs and B2B Evolve*. Gartner Research, Inc., 2004. <http://www.gartner.com>.
- [vdAtHKB03] W.M.P. van der Aalst, A.H.M. ter Hofstede, B. Kiepuszewski und A.P. Barros. Workflow Patterns. *Distributed and Parallel Databases*, Seiten 5–51, 2003.
- [W3C04] W3C. *Web Services Architecture*, 2004. <http://www.w3.org/TR/ws-arch/>.
- [Weg03] A. Wegmann. the systemic enterprise architecture methodology, 2003.
- [Wik06] Wikipedia. Flexibility — Wikipedia, The Free Encyclopedia, 2006. [<http://en.wikipedia.org/w/index.php?title=Flexibility&oldid=63058403>, accessed 12-July-2006].
- [Zac97] John A. Zachmann. Enterprise Architecture: The Issue of the Century. *Database Programming and Design Magazine*, 1997.
- [Zac99] J. A. Zachman. A Framework for Information Systems Architecture. *IBM Systems Journal*, 1999.