Bundesministerium
des Innern

Plattformunabhängigkeit von Fachanwendungen

Version 1.0

Leitfaden für Beschaffung, Entwicklung und Betrieb von plattformunabhängigen Fachanwendungen in der öffentlichen Verwaltung

www.kbst.bund.de

Koordinierungs- und Beratungsstelle der Bundesregierung für Informationstechnik in der Bundesverwaltung

KBSt

Schriftenreihe der KBSt
Band 91
Januar 2007

Schriftenreihe der KBSt

Band 91

Dieser Band wurde erstellt von der KBSt im
Bundesministerium des Innern (BMI)
und der EDS Business Solutions GmbH

Redaktion: EDS Business Solutions GmbH, Berlin

Interessenten erhalten die derzeit lieferbaren Veröffentlichungen der KBSt
und weiterführende Informationen zu den Dokumenten beim

Bundesministerium des Innern
Referat IT 2 (KBSt)
11014 Berlin

Homepage der KBSt: www.kbst.bund.de
Mailto: it2@bmi.bund.de

Plattformunabhängigkeit von Fachanwendungen

Leitfaden für Beschaffung, Entwicklung und Betrieb von plattformunabhängigen Fachanwendungen in der öffentlichen Verwaltung

Version 1.0

Berlin, Januar 2007

Herausgegeben vom
Bundesministerium des Innern

1 Einleitung

1.1 Motivation und Zielsetzung

Die IT-Verantwortlichen von Bund, Länder und Kommunen stehen vor der Herausforderung, dass sie nicht nur den Betrieb von komplexen IT-Infrastrukturen mit einer Vielzahl von unterschiedlichen Fachanwendungen zu verantworten haben, sondern darüber hinaus neuen Anforderungen bezüglich der Effizienz und informationstechnischen Vernetzung zur Realisierung prozessorientierter eGovernment-Lösungen der Behörden begegnen müssen.

Dabei geht es nicht nur um die Erweiterung bestehender Dienstleistungen, sondern zusätzlich um die Entwicklung vollständig neuer, komplexer Dienstleistungsangebote. Bei der Planung dieser neuen Konzepte sehen sich die IT-Verantwortlichen und Projektleiter in den einzelnen Behörden mit der anspruchsvollen Aufgabe konfrontiert, dass sie nicht auf der „grünen Wiese“ von Grund auf neu planen können, sondern dass die künftigen Angebote in die bestehende Infrastruktur eingepasst werden müssen.

Bei der Analyse der bestehenden IT-Infrastrukturen ergibt sich häufig das Bild, dass die einzelnen Komponenten durch klare Schnittstellenvereinbarungen modular und flexibel verbunden sind. Teilweise sind aber auch hochgradige Abhängigkeiten zwischen ihnen zu erkennen. Gerade solche Abhängigkeiten können aber unter Umständen bei der strategischen Weiterentwicklung einer IT-Landschaft erhebliche Probleme bereiten.

Leider ist es nicht möglich, pauschal eine Aussage darüber zu machen, welche Abhängigkeiten ein Problem bereiten oder welche nicht. Fast immer ist eine solche Einordnung stark an die konkrete Situation in der jeweiligen Behörde gebunden. Eine besondere Rolle spielt in diesem Kontext die Plattformabhängigkeit – die Abhängigkeit der Komponenten von den darunter liegenden Systemen und Umgebungen. Man spricht auch von der Ablaufumgebung der Anwendung.

An dieser Stelle setzt der vorliegende Leitfaden an. Er soll den IT-Verantwortlichen und IT-Projektleitern dabei helfen, ein Gespür für die technischen Abhängigkeiten in ihrem Umfeld und die daraus unter Umständen resultierenden Probleme zu entwickeln.

Dieser Leitfaden ist kein Masterplan für eine allgemeingültige, plattformunabhängige Landschaft von Fachanwendungen. Ein solcher Ansatz würde immer daran scheitern, dass die konkreten Anforderungen und Bedürfnisse in jeder Behörde zu unterschiedlich sind.

Vielmehr versteht sich der Leitfaden als Hilfe zur Selbsthilfe. Er gibt dem Leser das Rüstzeug, um Abhängigkeiten von bestehenden Fachanwendungen aufzudecken und hilft ihm dabei, Abhängigkeiten bei geplanten Fachanwendungen zu vermindern.

Die Grundlage für diesen Leitfaden sind die strategischen Zielsetzungen der IT- und der Software-Strategie des Bundes. Im Kontext Plattformunabhängigkeit sind für diesen Leitfaden dabei insbesondere die folgenden Zielsetzungen von Bedeutung:

- **Wirtschaftlichkeit**
  Ein gegebenes Maß an Aufgabenerfüllung bei geringst möglichem Mitteleinsatz sicherzustellen.
- **Investitionssicherheit**
  Die Nachhaltigkeit des Mitteleinsatzes zu gewährleisten.
- **Wiederverwendbarkeit**
  Die Möglichkeit schaffen, Modelle, Verfahren, Methoden, IT-Komponenten und andere IT-Systeme in unterschiedlichen Kontexten möglichst unverändert nutzen zu können.
- **Portabilität**
  Die Sicherung eines möglichst hohen Maßes an Unabhängigkeit, Computerprogramme in unterschiedlichen Ablaufumgebungen fehlerfrei ablaufen lassen zu können.
- **Softwarevielfalt**
  Die dauerhafte Verfügbarkeit alternativer Softwareprodukte am Markt, die für den gleichen Zweck einsetzbar sind.

Aus dieser Zielsetzung lässt sich die Forderung nach kontrollierter und gewollter Unabhängigkeit von Herstellern, Dienstanbietern, Betriebssystemen und Technologien ableiten. Eine solche Unabhängigkeit erhöht die Gesamtwirtschaftlichkeit des Einsatzes der Informationstechnik durch eine bereits im Design verankerte Austauschbarkeit von eingesetzten Produkten, was der Vergleichbarkeit der Angebote und dem Wettbewerb zugute kommt.

Die grundlegenden Begriffe und deren Relationen werden in der nachfolgenden Graphik dargestellt. Sie sind Gegenstand dieses Leitfadens und werden im weitern Verlauf dieses Dokuments näher betrachtet. Abhängigkeiten von Netzinfrastrukturen und Hardware können natürlich auch bestehen, werden aber im Rahmen dieses Leitfadens nicht näher betrachtet. Sie sind deshalb in der Graphik grau gekennzeichnet.

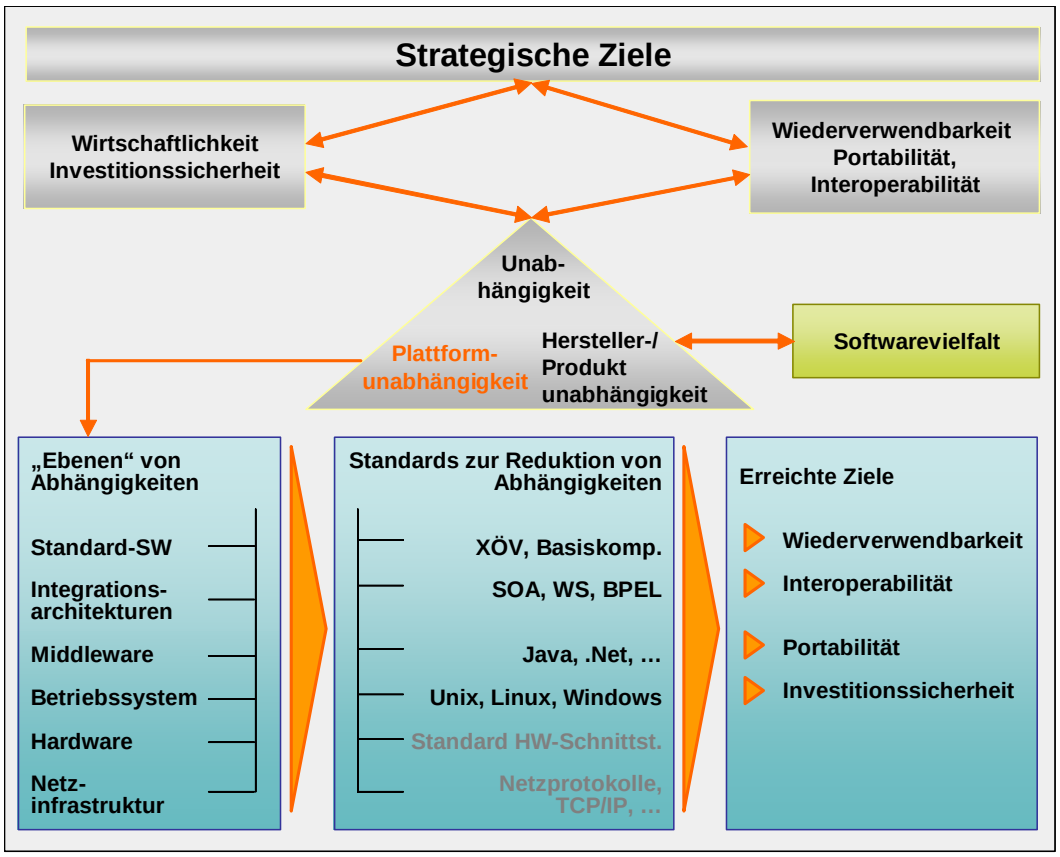


Abbildung 1: Grundlegende Begriffe und Beziehungen der Abhängigkeit

1.2 Über diesen Leitfaden

Auch wenn dieser Leitfaden von plattformunabhängigen Fachanwendungen handelt, beginnt er zunächst mit der Beschreibung von Abhängigkeiten.

Abhängigkeiten gibt es immer und sie können durchaus mit positiven Effekten verbunden sein. Erst wenn durch Abhängigkeiten notwendige Veränderungen behindert werden, sind sie ein Problem. Wichtig ist es daher, sich zunächst über die Abhängigkeiten im Klaren zu sein, um dann bewusst entscheiden zu können, an welchen Stellen Abhängigkeiten akzeptabel sind und an welchen Stellen sie kritisch werden können. Nur eine solche bewusste Entscheidung für oder gegen einzelne Abhängigkeiten führt letztlich zu Fachanwendungen, die in den wichtigen Punkten plattformunabhängig sind.

Das Kapitel 2 hilft dem Leser dabei, Abhängigkeiten auf verschiedensten Ebenen zu erkennen und zu bewerten. Da eine solche Bewertung situationsbedingt unterschiedlich ausfallen kann, ist es nicht möglich, eine vollständige Liste der kritischen Abhängigkeiten zu erstellen. Stattdessen wird dem Leser mit den Abhängigkeitsgraphen ein Werkzeug an die Hand gegeben, mit dem er selbst in der

Lage ist, in seiner speziellen Situation vorhandene Abhängigkeiten zu erkennen und zu bewerten.

Neben dem Erkennen von Abhängigkeiten spielt die Bewertung ihrer Auswirkungen eine große Rolle. Das Kapitel 2.2 widmet sich daher der Betrachtung der ökonomischen Auswirkungen, die durch Abhängigkeiten entstehen können. Dabei wird anhand konkreter Beispiele deutlich gemacht, dass Abhängigkeiten durchaus positive und negative Aspekte vereinen können. Zum Beispiel kann eine hoch integrierte IT-Infrastruktur die Produktivität der Anwender steigern, führt aber unter Umständen durch eine Vielzahl von Abhängigkeiten zu Einschränkungen der Handlungsspielräume bei der Veränderung der Infrastruktur.

Der Begriff der Plattformunabhängigkeit taucht in diesem Zusammenhang immer wieder auf und kann, je nach Aufgabenkontext, sehr unterschiedlich definiert werden. Je nach Definition werden Anwendungen daher häufig in plattformunabhängige und plattformabhängige Anwendungen unterteilt. Kapitel 3 stellt zunächst dar, dass eine solchen Unterteilung nicht allgemein gültig sein kann, da sie die Plattformunabhängigkeit auf die in der jeweiligen Definition enthaltenen Aspekte reduziert. Dieser Leitfaden fokussiert auf die Betrachtung der folgenden Ebenen und stellt verschiedene Ansätze vor, wie auf diesen Ebenen Plattformunabhängigkeit erreicht werden kann:

- Hardware
- Software
- Programmierumgebung
- Schnittstellen, Standards, Architekturen

In der Betrachtung von konkreten Situationen zeigt sich, dass Plattformunabhängigkeit in äußerst unterschiedlichen Ausprägungen vorliegen kann. Jede Behörde muss daher aus ihrer eigenen Situation heraus die Anforderungen an ihre zu erreichende Plattformunabhängigkeit definieren.

Die Kenntnis der technischen Ebenen bildet dabei die Basis für alle weiteren Betrachtungen, zum Beispiel im Rahmen einer Kosten-Nutzen Analyse. Dieser Leitfaden hilft dabei, die richtigen Fragen zu entwickeln, um eine technische Bewertung der Abhängigkeiten und ihrer Auswirkungen zu ermöglichen. Dazu werden konkrete Beispiele beschrieben und Hilfsmittel wie zum Beispiel eine Frageliste zur Verfügung gestellt. Anhand verschiedener Szenarien werden unterschiedliche Wege erläutert, die zu mehr Plattformunabhängigkeit führen und es wird deutlich gemacht, dass es den einen richtigen Weg nicht gibt.

Ein wichtiges Thema ist die Konsolidierung und Integration von IT-Infrastrukturen. Oft können Anwendungen heute nicht mehr isoliert betrachtet werden, sondern müssen als Bestandteil der gesamten IT-Infrastruktur gesehen werden.

Da insbesondere die Integration von Anwendungen eine Vielzahl von Abhängigkeiten generiert, wird dem Umgang mit diesen Abhängigkeiten ein eigener Ab-

schnitt gewidmet. Eine Integrationsarchitektur ist heute ein wichtiges Element einer modernen IT-Infrastruktur, die viele Vorteile bietet, deren Einsatz aber gut geplant werden muss, um nicht in ungewollte Abhängigkeiten zu geraten. Insbesondere den Themen Interoperabilität, offene Schnittstellen und offene Standards wird im Zusammenhang mit der Integration von Anwendungen Raum eingeräumt. Eine Betrachtung dieser Themen ist für die Reduzierung von Abhängigkeiten extrem wichtig. Da bei einer Integrationsarchitektur eine Anwendung nicht mehr isoliert betrachtet wird, erfordert der Aufbau einer solchen Architektur ein langfristig geplantes, strategisches Abhängigkeitsmanagement.

Während sich insbesondere das Kapitel 3 mit den grundsätzlichen Mechanismen und Möglichkeiten zur Erreichung von Plattformunabhängigkeit beschäftigt, geht das Kapitel 4 auf die Beschränkungen ein, die sich daraus ergeben, dass eine Anwendung in der Regel nicht auf der „Grünen Wiese“ entwickelt werden kann. Beschränkungen ergeben sich zum Beispiel aus der Organisation einer Behörde oder der vorhandenen Anwendungslandschaft.

Das Erkennen von Abhängigkeiten in solchen Landschaften ist ein zentraler Schritt für das Thema Plattformunabhängigkeit. Erst wenn klar ist wo, wie und warum Abhängigkeiten entstehen, ist es sinnvoll, nach Wegen zu suchen, um sie zu minimieren.

Die nachfolgende Abbildung stellt daher die wichtigsten Abhängigkeiten, die in einer IT-Infrastruktur existieren können, schematisch dar. Anschließend wird beschrieben, an welcher Stelle dieses Leitfadens mehr Information zu dem jeweiligen Thema zu finden ist.

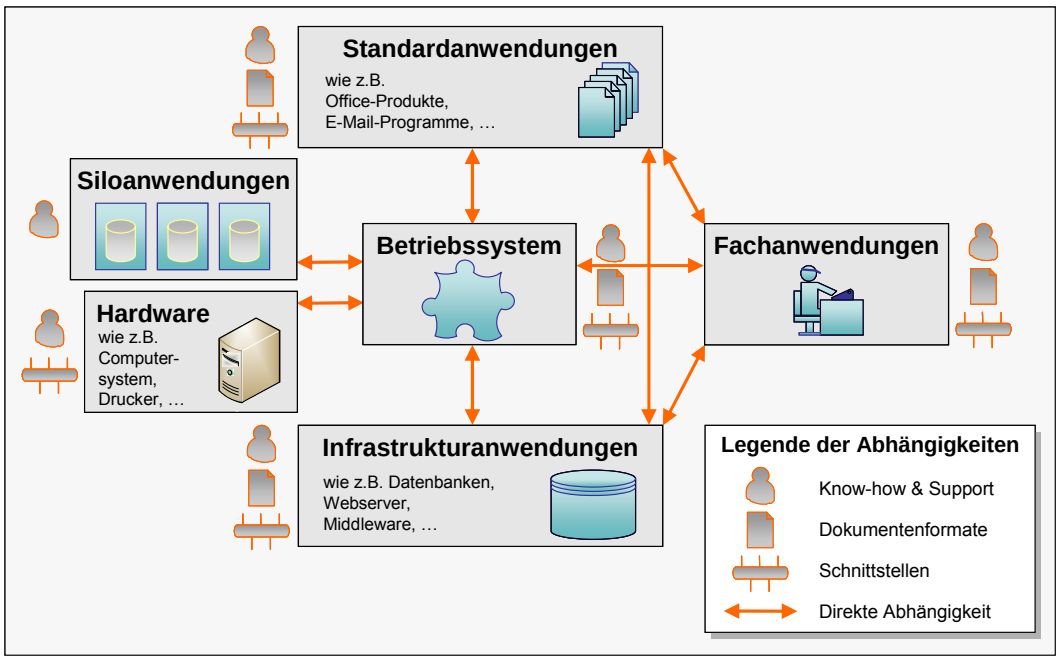


Abbildung 2: Abhängigkeit in einer IT-Infrastruktur

In dieser Abbildung ist gut zu erkennen, dass bei jeder Komponente der dargestellten IT-Infrastruktur eine Abhängigkeit zu Know-how und Support besteht. Im Wesentlichen handelt es sich dabei um eine eher organisatorische Abhängigkeit, die deshalb nicht im Fokus dieses Leitfadens liegt. Dennoch wird ein Aspekt dieser Abhängigkeit in Kapitel 2.1.5 dieses Leitfadens näher beleuchtet.

Auch die Abhängigkeit von Hardware, die in der Darstellung vereinfacht nur zum Betriebssystem dargestellt ist, ist in diesem Leitfaden kein Schwerpunkt. Auf sie wird in Kapitel 2.1.1 kurz eingegangen.

Die Abhängigkeit von Schnittstellen ebenso wie die Abhängigkeit von Dokumentenformaten ist sehr zentral und hat auf praktisch alle Komponenten der Infrastruktur Einfluss. Informationen zu diesen Abhängigkeiten finden sich daher an vielen Stellen dieses Leitfadens, vor allem aber in den Abschnitten 2.1.2, 2.1.3 und 3.2.4.

Die sehr häufig vorkommenden Abhängigkeiten vom Betriebssystem, der Infrastruktursoftware und der Standardsoftware werden schließlich in den Abschnitten 2.1.2, 2.1.3 und 2.1.4 vorgestellt.

Eines der zukünftigen Themen für die IT-Strategien der Behörden wird die Konsolidierung und Integration der jeweiligen IT-Landschaften sein. In der Abbildung werden dieses Thema und seine Konsequenzen für die Plattformunabhängigkeit nicht explizit dargestellt. Vielmehr sind von diesem Thema ausnahmslos alle in der Grafik dargestellten Komponenten betroffen. Der Abschnitt 0 befasst sich mit dem Komplex Plattformunabhängigkeit und Integration.

2 Problemstellung Plattformabhängigkeit

Die Entscheidung für den Kauf[1], die Auftragsentwicklung oder die Eigenentwicklung einer Fachanwendung – oder allgemeiner: eines Softwaresystems – wird heute nicht mehr alleine durch die fachlichen Anforderungen an die Software bestimmt.

Vielmehr fordern die Nutzer von modernen IT-Systemen ein hohes Maß an Interoperabilität zwischen den einzelnen Komponenten, sei es nun Hardware oder Software. Ein Vorteil solcher eng verzahnten Systeme besteht in einer höheren Benutzerfreundlichkeit und damit einer besseren Einsatz-Effizienz. Diese Vorzüge werden leider dadurch erkauft, dass in solchen integrierten Systemen eine Vielzahl von Abhängigkeiten unterschiedlicher Arten und Ausprägung entstehen können.

Dabei beeinflussen nicht alle Abhängigkeiten die IT-Systemlandschaft negativ. Abhängigkeiten werden erst dann problematisch, wenn sie verhindern, dass IT-Systeme die Veränderungen ihrer Umgebung nicht mehr oder nur unzureichend unterstützen oder notwendige Anpassungen einschränken oder sogar blockieren.

Das bedeutet, dass problematische Abhängigkeiten häufig zu unerwarteten und damit zu unplanbaren sowie zum Teil auch erheblichen Aufwendungen führen und darüber hinaus die möglichen Alternativen bei (strategischen) IT-Entscheidungen drastisch einschränken können. Deutlich geworden ist dies besonders in Migrationsprojekten, wo es darum ging, Windows NT – basierte IT-Infrastrukturen abzulösen. Die Plattformabhängigkeit von Fachanwendungen hatte hier in vielen Fällen maßgeblichen Einfluss auf die Entscheidungen hinsichtlich einer Umstellung auf eine alternative Plattform oder eine nachfolgende Generation der Windows-Plattform.

Um dies hier gleich vorweg zu nehmen, es handelt sich dabei um kein Spezifikum der Plattformabhängigkeit von Fachanwendungen von Windows oder anderen Microsoftprodukten. Die nachfolgend beschriebenen Beispiele können genauso auftreten, wenn die Abhängigkeit zu Linux oder anderer Plattformen besteht.

Besonders bitter waren die Fälle, wo die Umsetzung der strategischen Entscheidungen für einen Wechsel aufgrund der bestehenden Abhängigkeiten nicht vollzogen werden konnten. Dies betraf vor allem Behörden, die viele Fachanwendungen im Einsatz hatten und hier insbesondere jene, bei denen eine Vielzahl der Fachanwendungen als Standardanwendungen auf dem Markt beschafft wur-

[1] Die Beschaffung kann auch über Miete, Leasing oder einen Applikation Service Provider (ASP) erfolgen. Im Folgenden umfasst die Erwähnung der Möglichkeit der Beschaffung per Kauf auch immer die drei anderen Möglichkeiten.

den. Neben der Abhängigkeit von Windows und anderen Microsoft Produkten kam hier noch die Abhängigkeit von den Herstellern der Fachanwendungen hinzu. Dies bedeutete, dass in vielen Fällen bei einem Wechsel auf eine alternative Plattform neben der Ablösung der Basissoftwarekomponenten[1] auch eine große Zahl von Fachanwendungen (zum Teil bis zu 40 und mehr) neu zu beschaffen oder zu entwickeln gewesen wären. Dies war weder zeitlich noch finanziell für diese Behörden leistbar, so dass die bestehende Produktlinie zunächst fortgeführt werden musste. Realistisch gesehen kann nur ein langfristig strategisch ausgerichteter Plan zur Auflösung der bestehenden Plattformabhängigkeiten diesen Behörden für die Zukunft freie Entscheidungsmöglichkeiten eröffnen.

Aber auch in Behörden mit weniger Fachanwendungen im Einsatz führte die Plattformabhängigkeit dieser Fachanwendungen zu Zwangsentscheidungen, die nicht den strategischen Entscheidungen entsprachen und zudem zu erheblichen Mehraufwendungen führten. Konkret äußerte sich dies darin, dass keine vollständige Umstellung auf die alternative Plattform vorgenommen werden konnte sondern weiterhin Windows-Server, z.B. als Terminal-Server, betrieben werden müssen. Vor allem für die Arbeitsplatzcomputer (APC) musste außerdem eine Fortführung der bestehenden Produktlinie in Kauf genommen werden. Das heißt zusätzliche Lizenzen wurden notwendig, entsprechendes Know-how musste aufgebaut werden und Supportleistungen waren sicherzustellen. Ein prominentes Beispiel das sich hier einreihen lässt, ist das Migrationsprojekt der Stadt München, wo nach langer und intensiver Untersuchung am Ende doch nicht auf allen Arbeitsplatzcomputern (APC) ein Plattformwechsel, heißt, ein Wechsel nach Linux durchgeführt werden konnte. Aufgrund der Abhängigkeit einer Reihe wichtiger Anwendungen von Windows, mussten eine Reihe von APCs zunächst noch mit einem Windowsbetriebssystem ausgestattet bleiben.

Plattformabhängigkeit von Fachanwendungen hat in einigen Behörden auch dazu geführt, dass ein Wechsel der Officeplattform oder auch des verwendeten Datenbanksystems nicht durchgeführt werden konnte. Durch eine starke Integration der bestehenden Officeplattform in die Fachanwendung und der Nutzung proprietärer Funktionalitäten, hätten nach einem Wechsel auf eine alternative Officeplattform wichtige Funktionalitäten in den Fachanwendungen, wie z.B. die automatisierte Erstellung von Reports und Dokumentationen, den Anwendern nicht mehr zur Verfügung gestanden. Da die Behörden in den meisten Fällen keinen Einfluss auf eine nachträgliche Anpassung der Fachanwendung hatten, bestand nur die Wahl zwischen dem Wechsel der Fachanwendung (sofern überhaupt eine Alternative verfügbar war) oder der Fortführung der bestehenden Office-Produktlinie. Genauso lässt sich von einem Fall berichten, beim dem auf ein bestimmtes Datenbankmanagementsystem (DBMS) gesetzt wurde, was zunächst einmal der richtige Ansatz ist, denn der Betrieb vieler unterschiedlicher DBMS

[1] siehe auch Migrationsleitfaden Version 2.1, www.kbst.bund.de

erzeugt natürlich höhere Aufwendungen. Bei der Beschaffung und Entwicklung der benötigten Fachanwendungen wurde auch darauf geachtet, dass von diesen auch das ausgewählte DBMS unterstützt wurde. Worauf aber nicht geachtet wurde, war, dass keine proprietären Funktionalitäten dieses DBMS verwendet wurden. In fast allen Fachanwendungen wurden proprietäre Stored Procedure Funktionalitäten verwendet, so dass ein Wechsel des DBMS nur mit einer gleichzeitigen sehr aufwendigen Anpassung der Fachwendungen möglich gewesen wäre.

Damit wurde in den v.g. Fällen die Entscheidungsfreiheit der Behörden erheblich eingeschränkt bzw. sogar vollständig genommen. Zur Umsetzung der strategischen Zielsetzungen müssen jetzt erhebliche zusätzliche Anstrengungen unternommen werden, die bei einer rechtzeitigen Beachtung des Ziels Plattformunabhängigkeit hätten vermieden werden können.

Daher sollte vor jeder Einführung von neuen Fachanwendungen überprüft werden, ob durch die Einführung Abhängigkeiten im System entstehen, die nicht nur aktuell sondern auch auf lange Sicht zu Problemen führen können.

Im weiteren Verlauf dieses Kapitels werden technische Abhängigkeiten gruppiert und beschrieben und es wird dargestellt, wo bzw. wie diese entstehen können. Darüber hinaus wird gezeigt, wann Abhängigkeiten eine konkrete ökonomische Bedeutung erlangen können.

Diese Grundlagen erleichtern es dem Leser, eigenständig die wichtige Einordnung in problematische oder unproblematische Abhängigkeiten vorzunehmen. Dies ist nötig, da die Bewertung von Abhängigkeiten situationsbezogen ist und nicht statisch erfolgen kann. Das liegt daran, dass sich die Bedeutung von einzelnen Abhängigkeiten im Rahmen der Weiterentwicklung einer IT-Landschaft verändert. Die Konsequenz ist, dass eine allgemeingültige und eindeutige Liste, auf der alle problematischen Abhängigkeiten aufgeführt sind, nicht erstellt werden kann.

Aus diesem Grund kann dieses Kapitel keinen Anspruch auf Vollständigkeit erheben. Vielmehr ist sein Ziel, den Leser darin zu schulen, zusätzlich zu den beschriebenen, weitere Abhängigkeiten in seinem konkreten Verantwortungsbereich zu identifizieren und diese zu bewerten.

2.1 Technische Betrachtung

Abhängigkeiten im IT-Umfeld haben fast immer einen technischen Hintergrund und gehören so sehr zum Tagesgeschäft, dass sie schon fast nicht mehr auffallen.

Ein Beispiel: Aufgrund einer Gesetzesänderung muss für die neuen Aufgaben einer Behörde eine neue Fachanwendung angeschafft werden. Auf dem Markt sind verschiedene Produkte erhältlich, deren Einsatz denkbar wäre. Nun ist die Eignung der einzelnen Produkte für die konkreten Aufgaben zu prüfen.

Im ersten Schritt wird dabei die fachliche Eignung, d.h. die Abdeckung der fachlichen Anforderungen an das Produkt anhand von Broschüren oder Dokumentationen geprüft – soweit es möglich ist. Im Kontext solcher Dokumente werden früher oder später die so genannten „Systemanforderungen“ präsentiert. Das ist nichts Außergewöhnliches. Jede Software hat Systemanforderungen. Im Kontext dieses Leitfadens betrachtet, handelt es sich dabei aber um die offensichtlichen Abhängigkeiten der Anwendung.

Ausgehend von diesem Beispiel, lassen sich Abhängigkeiten in folgende technische Bereiche unterteilen

- Abhängigkeit von Hardware
- Abhängigkeit von Betriebssystemen
- Abhängigkeit von Infrastruktursoftware
- Abhängigkeit von Standardsoftware
- Abhängigkeit von technischem Know-how

Nicht für jeden der oben genannten Bereiche ist sofort ersichtlich, dass für Fachanwendungen Abhängigkeiten existieren bzw. entstehen können. Daher werden sie im weiteren Verlauf dieses Kapitels erklärt und mit Beispielen unterlegt. Ziel dieser Erklärung ist es, den Leser in seiner Wahrnehmung für Abhängigkeiten zu schärfen, um diese in seiner konkreten IT-Landschaft erkennen zu können. Zusätzlich wird in Abschnitt 2.1.6 ein Vorgehen vorgestellt, das dabei hilft, Abhängigkeiten in konkreten Situationen zu identifizieren.

2.1.1 Abhängigkeit von Hardware

Software kann direkt oder indirekt von Hardware abhängig sein.

Eine direkte Abhängigkeit bedeutet, dass die Software selbst bestimmte Hardware, seien es nun Computersysteme oder Peripheriegeräte, für die korrekte Ausführung benötigt. Ein gutes Beispiel dafür sind Betriebssysteme, die häufig nur auf bestimmten Hardwarearchitekturen lauffähig sind.

Eine indirekte Abhängigkeit von Hardware liegt dann vor, wenn nicht die Software selbst sondern eine von ihr benötigte Komponente von bestimmter Hardware abhängig ist. Das liegt zum Beispiel vor, wenn eine Software nur unter einem Betriebssystem lauffähig ist, das nur für eine bestimmte Hardware angeboten wird. Indirekte Abhängigkeit von Hardware ist nahezu immer vorhanden und wird im Normalfall nicht als problematisch betrachtet. Daher wird sie im weiteren Verlauf nicht näher betrachtet.

Insgesamt ist zu bedenken, dass eine Hardware-Abhängigkeit sich nicht nur auf Computer-Typen und Prozessoren reduziert. Auch klassische Peripheriegeräte wie Drucker, Scanner oder auch die Anforderung an die Ausstattung mit Haupt- oder Festplattenspeicher können Abhängigkeiten erzeugen.

Fachanwendungen sind nur in Ausnahmefällen direkt von konkreter Hardware abhängig. Solche Ausnahmefälle können dann vorliegen, wenn die Fachanwendung zum Beispiel spezielle Peripheriegeräte für die Benutzeridentifikation oder die Eingabe oder Ausgabe von Daten benötigt. So finden sich beispielsweise in der öffentlichen Verwaltung Arbeitsplätze, die Kartenlesegeräte für den Einsatz von digitalen Signaturen verwenden.

2.1.2 Abhängigkeit von Betriebssystemen

Die Erfolge, die das freie Betriebssystem Linux in den letzten Jahren erringen konnte, haben auch in der öffentlichen Verwaltung dazu geführt, dieses Betriebssystem einzusetzen.

So wurden mehrere Initiativen und Projekte wie zum Beispiel „bundestux.de“ oder „LiMux – Linux in München“ gestartet, um den Einsatz von Linux in öffentlichen Verwaltungen zu prüfen bzw. zu realisieren.

Eine einheitliche Erfahrung, die bei diesen Aktivitäten gewonnen wurde, ist, dass der Einsatz der meisten Fachanwendungen – ebenso wie der sonstiger Standardanwendungen auch – zwingend an die Verwendung bestimmter Betriebssysteme, meist Derivate von Microsoft Windows, gebunden ist.

Unabhängig von dieser konkreten Ausprägung kann aber allgemein festgestellt werden, dass Standardsoftware praktisch immer bestimmte Betriebssysteme, teilweise sogar spezielle Versionen der Systeme, voraussetzt. Diese Aussage kann sogar für Internet-Applikationen gelten, vor allem dann, wenn diese auf spezielle Internet-Browser zugeschnitten wurden, die nur für bestimmte Betriebssysteme existieren.

2.1.3 Abhängigkeit von Infrastruktursoftware

Bei Infrastruktursoftware handelt es sich um Software, die Dienste zentral für eine oder mehrere Anwendungen zur Verfügung stellt. Beispiele für Infrastruktursoftware sind Datenbank-Server, Mail-Server, Web- oder Applikationsserver.

Die durch Infrastruktursoftware bereitgestellten Dienste und Services folgen häufig internationalen Standards. Dadurch sollte eigentlich ausgeschlossen sein, dass Fachanwendungen beispielsweise in Abhängigkeit eines speziellen Datenbank-Servers geraten.

Allerdings ist für Infrastruktursoftware zu beachten, dass viele Anbieter Teile der Standards unterschiedlich auslegen bzw. bewusst um proprietäre Fähigkeiten erweitern. Dieses Phänomen trifft vor allem auf Datenbank- und Webserver zu.

Wenn Fachanwendungen auf Infrastruktursoftware mit proprietären Fähigkeiten aufbauen, bedeutet das nicht automatisch, dass sie auch in eine Abhängigkeit zu dieser Software gelangen. Entscheidend ist vielmehr, ob sie diese proprietären Funktionalitäten auch nutzen. So greifen Fachanwendungen – oder sonstige

Software – häufig auf zentrale Datenbank-Server, die nach dem Standard SQL arbeiten, zu. Solange eine Fachanwendung nur „echte“ SQL Kommandos und Befehle verwendet, ist sie nicht von einem speziellen Datenbank-Server abhängig. Nutzt sie aber auch nur eine einzige proprietäre Fähigkeit eines Datenbank-Servers, erzeugt sie damit eine Abhängigkeit von dieser Infrastruktursoftware.

2.1.4 Abhängigkeit von Standardsoftware

Kann eine Fachanwendung ohne eine bestimmte Standardsoftware, wie zum Beispiel IBM Lotus Notes oder Microsoft Excel, nicht ausgeführt werden, besteht für die Fachanwendung eine Abhängigkeit. Diese kann unterschiedlich stark ausgeprägt und damit auch unterschiedlich leicht erkennbar sein.

Zum einen gibt es Fachanwendungen, die bewusst auf Standardsoftware aufbauen. Dabei wird die Standardsoftware durch Konfiguration oder softwaretechnische Erweiterungen angepasst, damit diese den Anforderungen der öffentlichen Verwaltung genügt. Ein Beispiel dafür ist die Anwendung „DOMEA – Microsoft Edition“ von Open Text und Microsoft. Ziel dieser Fachanwendung ist es, das Microsoft Office Paket, speziell die Applikation Word, in DOMEA zu integrieren. Ein weiteres Beispiel wäre, wenn eine Fachanwendung z.B. für die Auswertung von Daten die API von OpenOffice.org nutzen würde, um die Daten nach Calc zu exportieren und auswerten zu können.

Darüber hinaus gibt es allerdings auch Fachanwendungen mit Abhängigkeiten von Standardsoftware, bei denen es sich auf den ersten Blick um eigenständige Programme handelt. Diese Fachanwendung sind zwar wie eigenständige Applikationen aufgebaut, nutzen aber Dienste und Funktionen von Standardsoftware. Dazu greifen sie über die proprietären Schnittstellen der jeweiligen Standardsoftware auf deren Funktionalität zu. Dazu gehören Applikationen, die auf Einzelplatz-Datenbanken wie Borland Paradox zugreifen, um ihre Daten zu speichern.

Bei vielen dieser Fachanwendungen ist ihre Abhängigkeit von Standardsoftware eine logische Konsequenz ihrer Entstehungsgeschichte. Häufig handelt es sich um eine aus Skript- bzw. Makroprogrammierung gewachsene Lösung für konkrete Aufgaben des Tagesgeschäfts. Nicht selten sind diese durch die Eigeninitiative einzelner Mitarbeiter entstanden – und dennoch oder gerade deswegen zum unverzichtbaren Bestandteil der täglichen Arbeit geworden. Weitere Ausführungen zum Thema "skriptende Power-User" finden sich in Abschnitt 4.1.4.2.

2.1.5 Abhängigkeit von technischem Know-how

Der Betrieb einer IT-Infrastruktur lässt sich nur dann sicherstellen, wenn auf ein tief greifendes technisches Wissen über ihre einzelnen Komponenten zugegriffen werden kann. Daraus folgt zwangsläufig, dass eine Abhängigkeit zwischen den Komponenten einer IT-Infrastruktur und dem technischen Know-how besteht.

Was bedeutet dies allerdings für die Abhängigkeit von Fachanwendungen? Auch hier ist wieder zwischen einer sehr deutlichen, direkten und einer indirekten Abhängigkeit zu unterscheiden.

Die direkte Abhängigkeit zwischen Fachanwendungen und technischem Know-how ist einfach zu erklären. Sehr vielen Fachanwendungen liegt eine so komplexe fachliche und technische Logik zu Grunde, dass ihr dauerhafter Betrieb nur durch entsprechend geschultes Personal sichergestellt ist. Hierbei wird beispielsweise Know-how für die Administration und Konfiguration der Anwendung ebenso wie für die Sicherung der Anwendungsdaten benötigt.

Die indirekte Abhängigkeit von Fachanwendungen zu Know-how ergibt sich erst auf den zweiten Blick. Besteht bei der Fachanwendung beispielsweise eine in Abschnitt 2.1.3 beschriebene Abhängigkeit von der Infrastruktur, ist zumindest ein Minimum an Know-how über diese Infrastruktur für den Betrieb der Anwendung nötig. Gleiches gilt natürlich auch für Abhängigkeiten von Hardware, Betriebssystemen oder Standardsoftware.

Die Abhängigkeit von Know-how gibt es in unterschiedlichen Ausprägungen. Grundsätzlich bezieht sie sich natürlich im ersten Schritt darauf, dass bei den eigenen Mitarbeitern mindestens ein gewisses Basiswissen vorhanden sein muss. Benötigen einzelne Fachanwendungen darüber hinaus sehr spezielles bzw. tiefes Know-how in einzelnen Bereichen, kann daraus eine Abhängigkeit zu externem Wissen resultieren. In letzter Konsequenz bedeutet dies eine Abhängigkeit von Support- bzw. Dienstleistungsverträgen.

2.1.6 Visualisierung von Abhängigkeiten

In diesem Kapitel wurden bislang verschiedene Abhängigkeiten dargestellt. Um solche Abhängigkeiten in der Praxis zu bestimmen, hilft es enorm, die Beziehungen zwischen den an einer Lösung beteiligten Infrastruktur-Komponenten zu visualisieren. Daraus resultierend können so genannte Abhängigkeitsgraphen entstehen.

Die Verwendung von Abhängigkeitsgraphen kann an einem einfachen Beispiel aus der Praxis nachvollzogen werden. In Abbildung 3 wird der Abhängigkeitsgraph für eine stark vereinfachte IT-Infrastruktur dargestellt. Der geneigte Leser kann die einzelnen Komponenten mit den realen Produkten aus seiner konkreten Umgebung füllen.

Sind die Komponenten in dem Graphen mit realen Produkten belegt, lassen sich an den Verbindungen zwischen den Komponenten die Abhängigkeiten ablesen. Darüber hinaus sind diese Verbindungen auch ein Indikator, durch den mögliche Freiheitsgrade bei der Produktauswahl erkannt werden können.

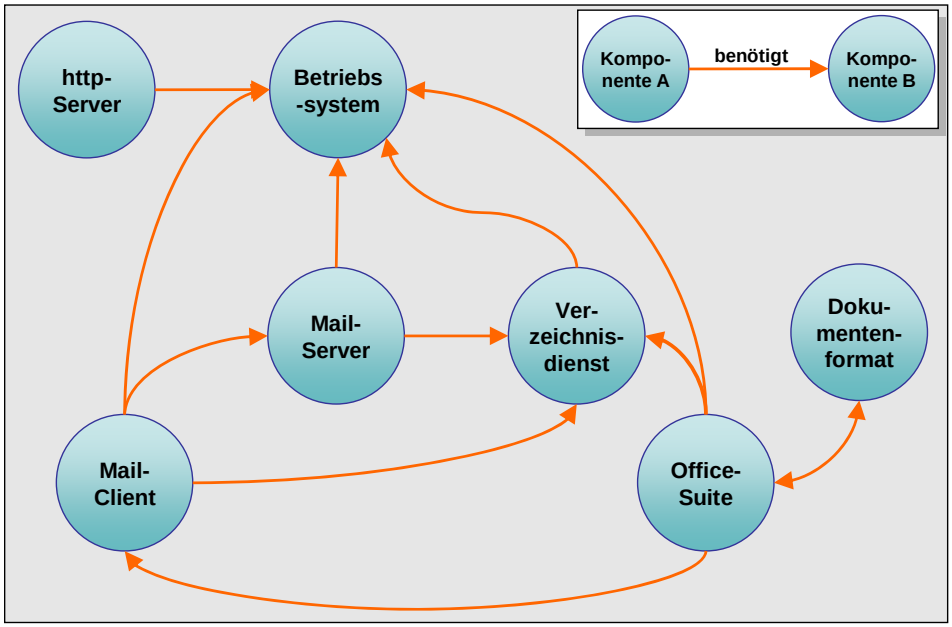


Abbildung 3: Abhängigkeitsgraph

Als Beispiel dafür soll die Beziehung zwischen einer Office-Suite und dem Dokumentenformat betrachtet werden. Fiktiv sei für die gewählte Umgebung angenommen, dass auf jeden Fall das Dokumentenformat „.doc“ von Microsoft Word bearbeitet werden muss.

Damit ist eine Komponente in dem Graphen fest vorgegeben, eine zweite kann daraus resultieren. Dazu ist zu überlegen, welche Office-Suiten das Dokumentenformat „Microsoft Word“ ebenfalls unterstützen. Nachfolgende Grafik stellt das symbolisch und beispielhaft dar.

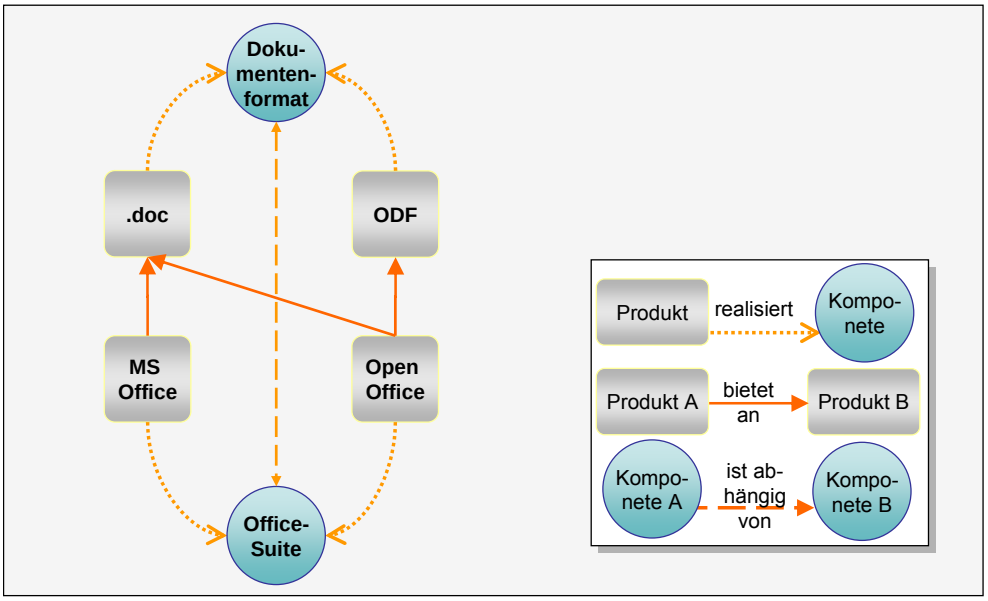


Abbildung 4: Beziehung Dokumentenformat zu OfficeSuite

Aus Abbildung 4 wäre zu erkennen, dass sowohl Microsoft Office als auch OpenOffice das Format „.doc“ unterstützen. Damit ergäbe sich aus der Beziehung Dokumentenformat zu Office-Suite, dass bei dieser Komponente eine Wahlmöglichkeit vorläge. Grundsätzlich unterstützt OpenOffice das Format „.doc“ allerdings gibt es schon noch kleine Unterschiede zwischen „.doc“-MS-Office und „.doc“-OpenOffice. Dies ist aber hier nicht das Thema, sondern die Nutzbarkeit solcher Graphen für Darstellung und Analyse von Abhängigkeiten und der Verfügbarkeit möglicher Alternativen soll deutlich gemacht werden.

Entsprechend können schrittweise die weiteren Komponenten mit Produkten belegt werden. Je nachdem, wie viele der Komponenten in einem solchen Abhängigkeitsgraphen fest vorgegeben sind, ist zu erkennen, wie die theoretische Wahlmöglichkeit zwischen verschiedenen Produktalternativen durch Abhängigkeiten verringert wird.

2.1.7 Fazit – Problematik der Plattformabhängigkeit aus technischer Sicht

Abschnitt 2.1 hat gezeigt, dass Abhängigkeiten nicht grundsätzlich ein Problem sind und in unterschiedlicher Ausprägung vorliegen können.

Um die Abhängigkeiten einer IT-Infrastruktur leichter zu erkennen, werden sie in verschiedene technische Klassen unterteilt. Ebenso wurde in diesem Abschnitt mit den Abhängigkeitsgraphen ein Werkzeug vorgestellt, mit dem Abhängigkeiten visualisiert werden können.

Ziel dieses Kapitel ist damit quasi der „erste Schritt“, nämlich den Leser für das Erkennen von Abhängigkeiten zu sensibilisieren. Hilfen für den zweiten Schritt, also die ökonomische Bewertung der erkannten Abhängigkeiten, werden in Abschnitt 2.2 gegeben.

2.2 Ökonomische Betrachtung

Die ökonomische Betrachtung von Abhängigkeiten bei Fachanwendung unterscheidet nicht zwischen problematischen und unproblematischen Abhängigkeiten. Das ist sinnvoll, da diese Einordnung nicht zuletzt durch die ökonomische Bewertung beeinflusst wird.

Die in diesem Abschnitt folgende ökonomische Betrachtung von Abhängigkeiten beschreibt die Auswirkungen, die aus den in Abschnitt 2.1 vorgestellten technischen Abhängigkeiten resultieren können. Die Konzentration auf die Auswirkungen ist sinnvoll, da beispielsweise die Abhängigkeiten von Betriebssystemen und die Abhängigkeit von Hardware aus wirtschaftlicher Sicht die gleichen Konsequenzen haben. Dazu kommt, dass für die ökonomischen Betrachtung der Abhängigkeiten weniger deren Ursachen als vielmehr deren Auswirkungen interessant sind.

Eine ökonomische Betrachtung von Abhängigkeiten im Rahmen dieses Leitfadens kann keine konkrete „Preistafel“ oder gar einen Wirtschaftlichkeitsrechner zum Resultat haben. Vielmehr werden im weiteren Verlauf dieses Kapitels anhand von Beispielen unterschiedliche Kostenblöcke dargestellt, die durch Abhängigkeiten entstehen können. Eine Quantifizierung dieser Kostenblöcke ist allerdings immer an konkrete Situationen gebunden und kann deshalb nicht pauschal erfolgen.

2.2.1 Eingeschränkte Alternativen

Die Beschaffung von Dienstleistungen und Material ist einer der größten Kostenblöcke in der öffentlichen Verwaltung. Um dem gesamten Wettbewerb eine Chance zu geben und den für den Auftraggeber wirtschaftlichsten Anbieter zu ermitteln, haben solche Beschaffungen i.d.R. durch öffentliche Ausschreibungen zu erfolgen und es sind begleitende Wirtschaftlichkeitsbetrachtungen durchzuführen. Im Rahmen der Ausschreibung sind die Anforderungen, die an die potentiellen Auftragnehmer gestellt werden, klar zu definieren. Im Rahmen der Wirtschaftlichkeitsbetrachtung ist zu untersuchen, ob die Wirtschaftlichkeit einer Lösung gegeben ist und, wenn mehrere Lösungsalternativen zur Verfügung stehen, welche die Wirtschaftlichste ist.

Werden zum Beispiel neue Computer für Sachbearbeiter angeschafft, müssen natürlich die von den Sachbearbeitern benötigten und vorhandenen Fachanwendungen weiterhin ohne Einschränkung nutzbar sein. Ist eine dieser Fachanwendungen von spezieller Hardware abhängig, beispielsweise einem integrierten Kartenlesegerät zum Zugriffsschutz, dann bedeutet dies ggf. eine Einschränkung hinsichtlich der Auswahl des kostengünstigsten Computers, die möglichen Mehrkosten für erweiterte Hardwareanforderungen müssen in Kauf genommen. Das im Beispiel damit erkaufte Mehr an Sicherheit bietet dagegen ggf. einen zusätzlichen wirtschaftlichen Vorteil, der in die Wirtschaftlichkeitsbetrachtung einfließen muss. Die Methodik der WiBe 4.0 (siehe www.kbst.bund.de) stellt hierfür die notwendigen Mittel für die Behörden zur Verfügung.

Wesentlich problematischer stellt sich das Fehlen von Alternativen bei Fachanwendungen dar, die von Betriebssystemen abhängig sind.

Dass Fachanwendungen vom Betriebssystem abhängig sind, ist auch in modernen IT-Infrastrukturen eher der Normalfall als die Ausnahme. Durch diese Abhängigkeit sind die Möglichkeiten, Änderungen oder Anpassungen der Infrastruktur durchzuführen, massiv beeinträchtigt.

Das wird insbesondere durch Projekte belegt, die sich mit der Migration auf das freie Betriebssystem Linux befassen. Die Projekte „LiMux“ (München), „WIENUX“ (Wien) und die Migration bei der BStU (Bundesbeauftragte für die Unterlagen des Staatssicherheitsdienstes der ehemaligen Deutschen Demokratischen Republik) sind gute Beispiele für solche Migrationen bei öffentlichen Verwaltungen. Die Erfahrungen aus diesen und vielen anderen Projekten besagen, dass durch die

Abhängigkeit vieler Fachanwendungen vom Betriebssystem der zeitliche und finanzielle Aufwand für die Migration dieser Fachanwendungen sehr groß sein kann. Aus wirtschaftlicher Sicht kann dies, dazu führen, dass keine vollständige Migration auf eine neue Zielplattform durchgeführt wird. Dann kann es wirtschaftlich begründet sein entweder einzelne Fachanwendungen, zumindest vorläufig, nicht auf das neue Zielsystem zu migrieren und, so lange diese Fachanwendungen noch genutzt werden, auch eine heterogene Betriebssystemlandschaft zu betreiben oder so lange auf eine Ablösung der vorhandenen Produktlinie des Betriebssystems zu verzichten, bis die Landschaft der Fachanwendungen mehr Unabhängigkeit aufweist.

Die beschriebenen Probleme können analog auftreten, wenn Fachanwendungen von Infrastruktursoftware oder Anwendungssoftware abhängig sind. Auch hier kann, wie bereits in der Einleitung zu Kapitel 2 dargestellt, ein geplanter Wechsel oder die Aktualisierung der Software, zu der die Abhängigkeit besteht, durch die Abhängigkeit eingeschränkt werden.

Zusammenfassend kann gesagt werden, dass durch Fachanwendungen, die in problematischen Abhängigkeiten zu Teilen der vorhandenen IT-Infrastruktur stehen, die Alternativen bei Veränderungen oder Weiterentwicklungen der IT-Umgebung eingeschränkt werden. Dies gilt vor allem dann, wenn es keine Möglichkeit gibt, entsprechende Fachanwendungen mit geringfügigen Aufwendungen in die neue Umgebung zu portieren.

2.2.2 Heterogener Know-how Bedarf

Durch die Abhängigkeiten, die Fachanwendungen zu unterschiedlichen Bereichen der IT-Infrastruktur haben können, kann eine Situation entstehen, die eine homogene Weiterentwicklung der IT-Landschaft massiv erschwert bzw. sogar völlig unmöglich macht.

Diese Problematik lässt sich an einem einfachen Beispiel verdeutlichen. Aus strategischen Gründen, wie z.B. der Verbesserung der Wirtschaftlichkeit oder der Schaffung von mehr Unabhängigkeit, soll die IT-Infrastruktur einer Behörde auf ein neues Betriebssystem umgestellt werden.

Bei einer einzelnen, dringend benötigte Fachanwendung ist allerdings der Wechsel des Betriebssystems unmöglich. Der Hersteller wird das neue Zielsystem nicht unterstützen. In dieser Situation ergeben sich verschiedene Alternativen, die sich aber zu drei Hauptalternativen zusammenfassen lassen:

- Der Wechsel des Betriebssystems wird nicht durchgeführt.
- Die betroffene Fachanwendung wird ausgetauscht.
- Um die betroffene Fachanwendung weiter betreiben zu können, wird das alte Betriebssystem zusätzlich bereitgestellt.

Die Realisierung der zweiten Alternative bedeutet, dass zusätzlich zu der Migration der Betriebssysteme die Migration der betroffenen Fachanwendung durchzuführen ist.

Wird, wie im dritten Fall vorgeschlagen, die betroffene Fachanwendung in der alten Umgebung weiter betrieben, sind viele unterschiedliche Aspekte zu beachten. Dabei ist es vor allem wichtig, dass die Nutzerzahlen und die Nutzungshäufigkeit der Anwendung berücksichtigt werden.

Der Lösungsansatz, das alte Betriebssystem zusätzlich zu dem neuen anzubieten, bedeutet für die IT-Infrastruktur, dass zumindest eine gewisse Heterogenität der Infrastruktur in Kauf genommen wird. Beispiele für diese Heterogenität sind in fast allen Migrationsprojekten in der öffentlichen Verwaltung zu finden, wo Fachanwendungen eingesetzt werden. Insbesondere Kommunalverwaltungen sind hiervon betroffen, da dort i.d.R. sehr viele Fachanwendungen im Einsatz sind.

Die Entscheidung für eine solch heterogene Lösung hat weit reichende Konsequenzen. So muss für jeden Bestandteil einer heterogenen Lösung der Betrieb und die Wartung sichergestellt werden. Auch erhöhen sich unter Umständen die Know-how Anforderungen – und damit der Schulungsbedarf – an die Mitarbeiter, die mit solchen heterogenen Lösungen arbeiten müssen. Dabei ist es unerheblich, ob diese Mitarbeiter fachliche oder technische Aufgaben an diesen Systemen wahrnehmen.

Im Resultat müssen also für jeden Bestandteil einer heterogenen Lösung Ressourcen für den Betrieb, die Unterstützung und ähnliches bereitgestellt werden. Der dadurch anfallende Mehraufwand kann sowohl zu einer Mehrbelastung einzelner Mitarbeiter als auch zum Bedarf, neue Mitarbeiter einzustellen, führen.

2.2.3 Herstellerabhängigkeit

Werden in einer IT-Infrastruktur Fachanwendungen eingesetzt, die abhängig von Komponenten dieser Struktur sind, kann es in der Folge zu einer Herstellerabhängigkeit kommen.

Auch dieser Sachverhalt lässt sich leicht an einem Beispiel verdeutlichen: Die gesamte Softwarelandschaft einer Behörde ist hoch integriert und dadurch auf den Einsatz eines speziellen Betriebssystems zugeschnitten. Das gilt auch für Fachanwendungen, die im Auftrag der Behörde bzw. von ihr selbst gefertigt wurden. Eine Konsequenz dieses Systemaufbaus ist, dass ein Wechsel des Betriebssystems – wenn überhaupt – nur mit hohem zeitlichem und finanziellem Aufwand möglich ist. Dadurch ist die IT-Infrastruktur der Behörde von vielen Entscheidungen, die der Hersteller des Betriebssystems trifft, abhängig.

Herstellerabhängigkeit ist immer dann eine mögliche Folge, wenn Fachanwendungen abhängig von einzelnen Komponenten der Infrastruktur sind. Das gilt für alle in Abschnitt 2.1 vorgestellten technischen Abhängigkeiten gleichermaßen.

Die Abhängigkeit von einem Hersteller kann dabei unterschiedliche Ausprägungen haben. Gerade gegenüber großen IT-Anbietern sind die Möglichkeiten, auf die Release-Politik und den Funktionsumfang von Anwendungen Einfluss zu nehmen, verschwindend gering. Wesentlich direktere Auswirkung der Herstellerabhängigkeit haben allerdings die Lizenz- und Preispolitik der Hersteller. Entscheidungen, die der Hersteller hier trifft, können unter Umständen direkt haushaltswirksam werden und damit massiv in die Haushaltsplanung eingreifen.

Gerade im Bereich der Herstellerabhängigkeit zeigt sich, wie klein der Schritt von einer unproblematischen zu einer problematischen Abhängigkeit ist. Das Verhalten von externen Anbietern kann nur begrenzt durch die öffentliche Verwaltung beeinflusst werden. Allerdings können solche Entscheidungen von Herstellern massiv Auswirkungen auf die IT-Infrastruktur einer Behörde haben.

2.2.4 Fazit zur ökonomischen Betrachtung

Insgesamt bestätigt sich bei der Betrachtung der ökonomischen Auswirkung die Notwendigkeit, die Plattformunabhängigkeit in ihren Ausprägungen zu analysieren sowie Mittel und Wege zu deren methodischen Bewertung bei den Entscheidungen im Kontext von Anwendungen zu finden.

Ohne eine solche Bewertung ist es nicht möglich, Grundsatzaussagen über die positiven oder negativen Auswirkungen von Abhängigkeiten zu machen. Praktisch alle Abhängigkeiten können als „Fluch und Segen“ betrachtet werden. So steigert eine hoch integrierte IT-Infrastruktur im Normalfall die Produktivität für die Anwender dieser Umgebung. Allerdings kann die hohe Integration schnell zu einem Problem werden, wenn einzelne Komponenten z.B. aufgrund fehlender Qualität ausgetauscht werden müssen und nicht können.

An dieser Stelle gilt die vorab genannte Faustregel:

> *Abhängigkeiten haben dann einen zu großen Grad erreicht, wenn sie eine gewollte Veränderung und Verbesserung verhindern. Dies gilt sowohl für den Einsatz von einzelnen Komponenten als auch für den Wechsel von ganzen Schichten einer IT-Architektur.*

Es sei noch einmal darauf hingewiesen, dass die Erkennung und Bewertung von Abhängigkeiten keine statische Aufgabe ist. Darüber hinaus verlangt die ständige Überprüfung der IT-Infrastruktur auf problematische Abhängigkeiten intime Kenntnisse der Umgebung und ein hohes Maß an Feingefühl für zukünftige Entwicklungen.

Zusätzlich gibt es noch Maßnahmen, die dazu geeignet sind, Probleme zu minimieren und Lösungsmöglichkeiten offen zu halten, um auf entstehende Abhängigkeiten zu reagieren. Diese Maßnahmen werden in Abschnitt 3.2 vorgestellt.

3 Gestaltungsmerkmale plattformunabhängiger Architekturen

Das Thema Plattformunabhängigkeit erreichte mit den ersten Versionen der Programmiersprache Java von SUN Microsystems erstmals ein Massenpublikum. Die Aussage „Write once, run everywhere“[1] weckte sowohl bei Programmierern als auch bei IT-Verantwortlichen die Hoffnung auf einen deutlichen Zuwachs an Flexibilität beim Einsatz von Anwendungen.

Die Erfahrungen, die seit dem mit dem Thema Plattformunabhängigkeit gemacht wurden, haben ergeben, dass Plattformunabhängigkeit viele unterschiedliche Facetten hat. Daher besteht die Aufgabe dieses Kapitels darin, den Begriff „Plattformunabhängigkeit“ aus Sicht dieses Leitfadens abzugrenzen und zu definieren. Auf Basis dieser Definition werden Software-Architekturen sowie Frage- und Checklisten vorgestellt, die dabei unterstützen können, eine entsprechende Plattformunabhängigkeit in der Praxis zu erzielen.

Abschließend befasst sich das Kapitel mit den Aufgaben, die bei der Integration von mehreren Fachanwendungen bzw. ganzen IT-Infrastrukturen zu bewältigen sind, um trotz der Integration ein hohes Maß an Plattformunabhängigkeit zu erhalten.

Es sei aber bereits an dieser Stelle erwähnt, dass Architekturen oder Fragelisten in der Praxis bei den Bemühungen, plattformunabhängige Fachanwendungen zu erhalten, durchaus helfen können, garantieren können sie die Plattformunabhängigkeit jedoch leider nicht. Dazu gibt es bei konkreten Problemstellungen in der Praxis zu viele Details, bei denen falsche Lösungsansätze auch prinzipiell plattformunabhängige Architekturen an eine bestimmte Plattform binden können.

Damit gilt bei der Vermeidung von Plattformabhängigkeiten das Gleiche wie bei der in Kapitel 2 beschriebenen Suche nach Abhängigkeiten: Die in diesem Leitfaden vorgestellten Architekturen und Fragelisten sind Hilfsmittel, die immer auch durch fachliche und technische Expertise ergänzt werden müssen.

3.1 Ausprägungen der Plattformunabhängigkeit

Sucht man im Internet nach Definitionen und Erklärungen zum Thema „Plattformunabhängigkeit“ ergibt sich ein weites Bild von dem, was als „plattformunabhängig“ bezeichnet wird.

Sehr viele der entsprechenden Webseiten drehen sich um so unterschiedliche Themen wie plattformunabhängige Dateiformate, Webseiten, Programmiersprachen und vor allem natürlich plattformunabhängige Anwendungen. Trotz der Brei-

[1] Sinngemäß: „Einmal programmieren, überall ausführen.“

te der Aussagen und aller Unterschiede im Detail haben diese Artikel, Definitionen und Aussagen einen gemeinsamen Kern.

Die freie Internet Enzyklopädie Wikipedia beschreibt Plattformunabhängigkeit beispielsweise als „die Eigenschaft eines Programms, auf verschiedenen Computersystemen mit Unterschieden in Architektur, Prozessor, Betriebssystem, etc. lauffähig zu sein."

Diese Definition enthält, etwas versteckt, diesen gemeinsamen Kern. In der Theorie mag es eine Plattformunabhängigkeit geben, die sich über alle denkbaren Computersysteme erstrecken kann. In der Praxis gilt Plattformunabhängigkeit allerdings schon dann als erreicht, wenn mehrere verschiedene Computersysteme bzw. Betriebssysteme unterstützt werden. In der englischsprachigen Fachliteratur wird dementsprechend auch häufig der die Realität besser beschreibende Ausdruck „cross-platform" anstelle von „platform independent" verwendet.

Die Definition von Wikipedia suggeriert noch etwas anderes: Ein Programm ist plattformunabhängig oder es ist es nicht. Dieser Leitfaden zeigt allerdings, dass eine binäre Wertung zu kurz gegriffen ist und es verschiedene Ausprägungen von Plattformunabhängigkeit gibt. Diese Ausprägungen sind ein Resultat von vielen Einzelmaßnahmen, die auf unterschiedlichen Ebenen eines Computersystems durchgeführt werden.

Erst unter Betrachtung der einzelnen Ebenen kann eine Aussage über die Ausprägung der Plattformunabhängigkeit einer Fachanwendung getroffen werden. Vorweg muss hierzu auf das Zusammenspiel der Ebenen ausführlicher eingegangen werden.

Das erklärte Ziel der Plattformunabhängigkeit besteht darin, durch Austausch der „darunter liegenden" Schichten[1] einer existierenden Umgebung profitieren zu können. Diese Definition ist grundsätzlich generisch, d.h. nicht von einer konkreten Plattform abhängig und der erwähnte Nutzen kann sich aus unterschiedlichen Faktoren ergeben. Im einfachsten Fall ergibt er sich durch die Erneuerung und Modernisierung einer darunter liegenden Schicht unter Einsatz höherwertiger oder preisgünstigerer Komponenten.

Die Begriffe einer „darunter liegenden" und einer „darüber liegenden" Schicht spielen für die weiteren Überlegungen eine wesentliche Rolle. Aus dem Verständnis einer Plattform als einer aus dem Blickwinkel der betrachteten Komponente ((Software)Service, (Fach-)Anwendung, Basis-System[2] usw.) stets „darunter liegenden" Schicht ergeben sich zwei wichtige Konsequenzen:

---

1 Die Existenz eines beliebigen Schichtenmodells vorausgesetzt

2 Nicht zu verwechseln mit den Basiskomponenten aus SAGA 2.1 oder dem Einer-für-Alle-System (EfA-System), der Begriff, der zukünftig verwendet wird.

1. Plattformunabhängigkeit handelt mit einer Hierarchie von Komponenten und Zusammenhängen, bzw. lässt sich auf eine solche zurückführen, und
2. die Kopplung, bzw. Entkopplung der einzelnen Ebenen bestimmt das Maß der erreichten Ab-, bzw. Unabhängigkeit innerhalb des Gesamtsystems

Die nachfolgende Abbildung visualisiert beispielhaft das hierarchische Plattform-Modell. Dieses Modell ist nicht mit den in einem späteren Abschnitt vorgestellten n-Schichten-Architekturen zu verwechseln. Hier wird der Fall betrachtet, dass eine Anwendung ihre Funktionalität oder Teile davon als Service bereitstellt. Die Integration von Services in Fachanwendungen oder weiteren Services wird ausführlich in späteren Abschnitten dieses Leitfadens behandelt.

Aus dem Blickwinkel des beispielhaft aufgeführten Geschäftsprozesses und des dazugehörigen (Software)Services stellt die darunter liegende Ebene, also in diesem Fall ein Service-Bus, bzw. der Service-Broker die Plattform dieses Services dar.

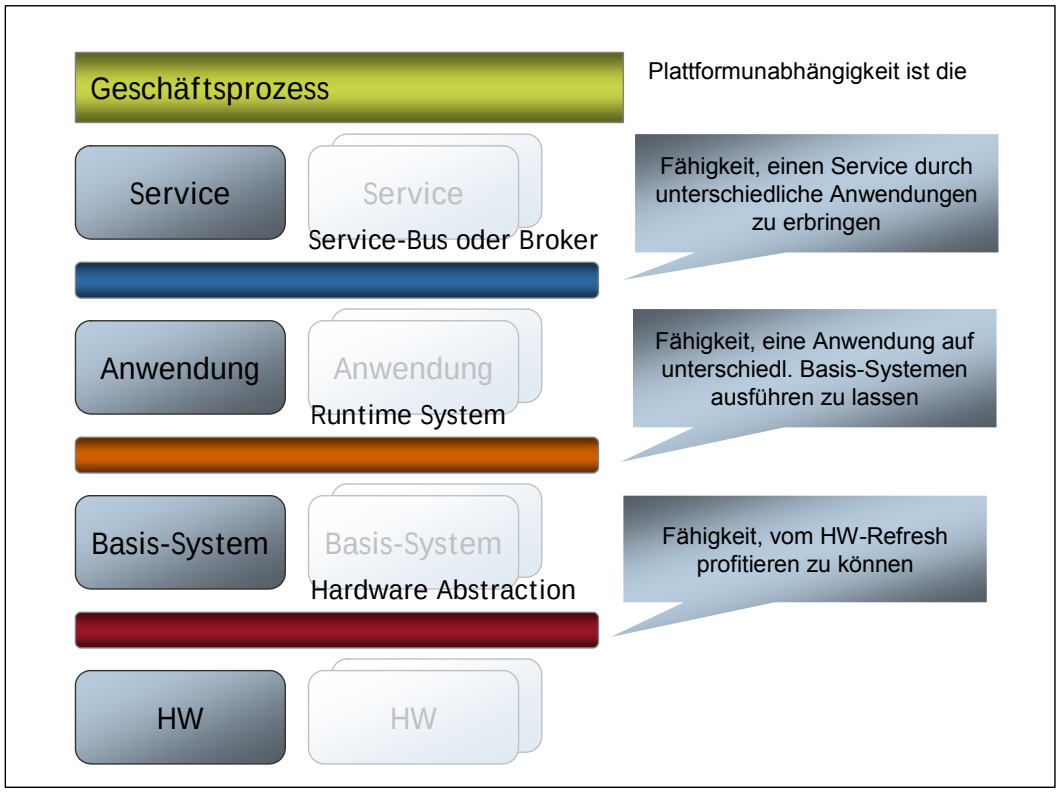


Abbildung 5: Hierarchisches Plattformmodell

Betrachtet aus dem Blickwinkel der Fachanwendung bildet die direkt darunter liegende Ebene ihre Plattform. Falls (wie in der obigen Abbildung dargestellt) die

Anwendung durch ein Runtime-System vom Basis-System[1] entkoppelt wird, stellt das Runtime-System die Plattform dar.

Die Entkopplung von Ebenen resultiert primär aus den Grundsätzen der modernen Software- und Systementwicklung, stellt jedoch in diesem Kontext gleichzeitig den Maßstab zur Beurteilung der Plattformunabhängigkeit dar. In den nachfolgenden Abschnitten werden die in diesem Leitfaden verwendeten Ausprägungen der Plattformab- bzw. –unabhängigkeit entsprechend diesem Maßstab definiert.

Tabelle 1: Ausprägungen der Plattformunabhängigkeit

0. Vollständige Plattformunabhängigkeit	Eine *vollständige Plattformunabhängigkeit* eines Geschäftsprozesses liegt dann vor, wenn auf allen Ebenen des oben aufgeführten hierarchischen Plattformmodells (siehe Abbildung 5) die Ebenen jeweils plattformunabhängig sind - Idealzustand.
1. (Weitgehende) Plattformunabhängigkeit	Eine **Fachanwendung** kann als (*weitgehend) plattformunabhängig* bezeichnet werden, wenn sie von dem Basis-System auf eine standardisierte Art und Weise entkoppelt ist, auf mehreren Basis-Systemen ausgeführt werden kann und somit von einem Austausch dieser profitiert.
2. Eingeschränkte Plattformunabhängigkeit	Eine **Fachanwendung** kann als *eingeschränkt plattformunabhängig* bezeichnet werden, wenn sie zwar vom zugehörigen Basis-System durch eine Laufzeitumgebung entkoppelt ist, jedoch aus unterschiedlichen Gründen nur auf einem Basis-System ausgeführt werden kann. Dies ist z.B. der Fall, wenn die verwendete Programmier- und Laufzeitumgebung nur für ein Basis-System verfügbar ist.
3. Keine Plattformunabhängigkeit	Eine **Fachanwendung** ist *plattformabhängig*, wenn sie nicht von dem darunter liegenden Basis-System entkoppelt ist und dadurch nicht unabhängig von seiner konkreten Ausprägung ausgeführt werden kann.

[1] Nicht zu verwechseln mit den Basiskomponenten aus SAGA 2.1 oder dem Einer-für-Alle-System (EfA-System), der Begriff, der zukünftig verwendet wird.

Die Feststellung der Ausprägung von Plattformunabhängigkeit spielt insbesondere im Rahmen der Wirtschaftlichkeitsbetrachtung eine wichtige Rolle, nämlich dann, wenn es darum geht eine sachgerechte Bewertung des entsprechenden Kriteriums durchzuführen. Im Detail wird dieser Aspekt im Kapitel 4.2.2 betrachtet.

In den nachfolgenden Abschnitten wird auf die Eigenschaften der aufgeführten Begriffsbestimmung näher eingegangen.

3.1.1 „Optimale“ Plattformunabhängigkeit

Eine vollständige Plattformunabhängigkeit ist nicht notwendigerweise ein um jeden Preis zu verfolgendes Ziel, obwohl sie das genannte Kriterium Plattformunabhängigkeit vollständig erfüllt. Die Begründung hierfür liegt primär in der Wirtschaftlichkeit, ganzheitlich betrachtet.

In den einleitenden Kapiteln ist bereits der Aspekt einer „positiven“ Abhängigkeit angesprochen und diskutiert worden. Eine positive Abhängigkeit resultiert im Regelfall aus der durch die Hersteller oder Lösungsanbieter vollzogenen Integration von Einzelteilen eines Gesamtsystems und sorgt durch eine höhere „Vorfertigungstiefe“ für eine bessere Wirtschaftlichkeit. Gesamtsystem in diesem Zusammenhang kann eine einzelne Fachanwendung sein, kann aber z.B. auch eine Fachanwendung plus das dazugehörige Datenbanksystem und ggf. weiteren notwendigen Standardtools sein. Gesamtsystem also in dem Sinne, dass nur durch die Gesamtheit aller Einzelteile die Vollständigkeit der benötigten Funktionalitäten gegeben ist. Allerdings gelten auch für die Abhängigkeiten zwischen den Einzelteilen des Gesamtsystems, dass sie einen gewünschten Wechsel der darunter liegenden Plattformen nicht verhindern dürfen.

Die Plattformunabhängigkeit muss einen wirtschaftlichen Nutzen bringen. Dieser ergibt sich – aus Sicht einer Fachanwendung – durch den positiven Effekt der Austauschbarkeit von darunter liegenden Schichten des Basis-Systems[1]. Dieser Effekt ist darin begründet, dass die Refresh-Zyklen bei Hardware und hardwarenahen Softwareprodukten wie z.B. Betriebssystemen im Regelfall wesentlich schneller sind als bei den Fachanwendungen, die häufig mehrere Generationen von Hardware oder Betriebssystemen überstehen. Darüber hinaus ist Plattformunabhängigkeit dann wichtig, wenn von vornherein feststeht, dass eine Fachanwendung von Anfang an in unterschiedlichen Umgebungen eingesetzt werden (z.B. EfA-Anwendungen).

Die im Einzelfall optimale Plattformunabhängigkeit ergibt sich somit durch ihre Wirtschaftlichkeit. Bei der Wirtschaftlichkeitsbetrachtung muss darauf geachtet werden, dass die Aufwendungen, die zur Erreichung der Plattformunabhängigkeit

[1] Nicht zu verwechseln mit den Basiskomponenten aus SAGA 2.1 oder dem Einer-für-Alle-System (EfA-System), der Begriff, der zukünftig verwendet wird.

benötigt werden, einen positiven Saldo ergeben. Um dieses Ziel erreichen zu können, muss die Plattformunabhängigkeit methodisch in die Wirtschaftlichkeitsbetrachtung eingebettet werden[1].

Aus diesen Überlegungen lässt sich das Prinzip der Verhältnismäßigkeit ableiten:

> *Es muss darauf geachtet werden, dass der Aufwand für die Realisierung der Plattformunabhängigkeit, nicht den Aufwand wesentlich übersteigt, der für den Austausch der Fachanwendung bei einem Plattformwechsel notwendig ist.*

3.1.2 Hardwarebezogene Plattformunabhängigkeit

Aus Sicht dieses Leitfadens werden alle physikalischen Elemente eines Computersystems, eines Netzwerks oder eines Speichersystems als Hardware bezeichnet. Allerdings: Auch wenn es eine Vielzahl an Hardware-Varianten geben kann, wird Hardware nur am Rande betrachtet.

Im Normalfall wird die Hardware eines Computersystems durch das Betriebssystem und somit das Basis-System gekapselt. Durch eine in der Vergangenheit stets schnell erfolgte Standardisierung der Hardware-Schnittstellen ist die überwiegende Mehrheit der heutigen Fachanwendungen nicht hardwarenah programmiert, sondern greift über das Betriebssystem auf die Hardware zu. Dadurch sind Änderungen der Hardware eines Computersystems für die Fachanwendungen unsichtbar und haben höchstens Einfluss auf ihre Leistungsfähigkeit.

Von dieser Aussage gibt es zwei Ausnahmen: Wird trotz gleich bleibendem Betriebssystem die zugrunde liegende Hardware-Architektur gewechselt, soll zum Beispiel statt eines Intel Pentium ein RISC-Prozessor verwendet werden, ist praktisch sicher mit Auswirkungen auf betriebssystemnahe Anwendungen zu rechnen. Zu diesen Auswirkungen kann zum Beispiel die erneute Kompilation[2] des Quelltextes der Anwendung gehören. Auch wenn solche Hardware-Wechsel durch geeignete Software-Architekturen oder Programmiersprachen reduziert und teilweise sogar vermieden werden können, ist damit zu rechnen, dass ein gewisser Anpassungsbedarf bei den Fachanwendungen auftritt.

Die zweite Ausnahme sind diejenigen Fachanwendungen, die bewusst hardwarenah programmiert wurden. Das kann nötig sein, wenn beispielsweise aus Sicherheitsgründen die Nutzung bestimmter Hardware, z.B. aufgrund ihrer Zertifizierung oder Zulassung, verpflichtend ist. In einem solchen Fall liegt dann natür-

---

1 siehe Kapitel 4.2.2 Seite 117

2 Bei diesem Vorgang wird aus den bestehenden Quelltexten einer Fachanwendung die eigentlich benötigte ausführbare Datei erstellt.

lich eine Abhängigkeit von der Hardware vor, die durch geeignete Softwarearchitekturen unter Umständen reduziert, aber kaum beseitigt werden kann.

3.1.3 Unabhängigkeit vom Basis-System[1]

In diesem Leitfaden wird der Begriff Basis-System als Zusammenfassung der basistechnologischen Komponenten zur Ausführung von Fachanwendungen verwendet. In den vergangenen Dekaden hat sich sowohl der Begriff des Betriebssystems als auch die Integrationstiefe grundlegend verändert, so dass es nicht mehr angebracht ist, nur das Betriebssystem als Umgebung zur Ausführung einer Anwendung zu sehen.

Aus Sicht dieses Leitfadens umfasst ein Basis-System z.B. folgende Komponenten:

- Betriebssystem
- Datenbankmanagementsystem (z.B. RDBMS)
- Berechtigungsmanagementsystem (z.B. LDAP)
- Messaging-System, wenn es als Engine verwendet wird
- Office-Anwendungen (z.B. Tabellenkalkulation)
- Ggf. weitere Komponenten

Das Basis-System repräsentiert alle Ressourcen, die eine Fachanwendung zu ihrer Ausführung und zum Management ihrer Daten benötigt. Aus diesem Blickwinkel heraus ist es ebenfalls unangebracht, Abhängigkeiten von den einzelnen aufgeführten Komponenten des Basis-Systems isoliert zu betrachten. Eine Fachanwendung ist entweder vom Basis-System im Ganzen durch Schnittstellen und Laufzeitumgebungen entkoppelt oder sie ist in unterschiedlicher Form von ihm abhängig.

3.1.4 Die Anwendung und ihre Laufzeitumgebung

Wie bereits in der Einleitung zu diesem Kapitel erwähnt, hat die Entwicklung der Programmiersprache Java einen großen Einfluss auf das Thema Plattformunabhängigkeit gehabt. Daher ist es nicht weiter verwunderlich, dass Programmiersprachen und damit auch Programmierumgebungen eine wichtige Rolle für plattformunabhängige Fachanwendungen spielen.

Vorhanden als Entwicklungs- und Laufzeitumgebung ermöglicht Java eine bis auf wenige Ausnahmen vollständige Entkopplung (Kapselung) der Fachanwendung vom Basis-System. Dieses Prinzip wird durch mehrere Technologien verwirklicht,

[1] Nicht zu verwechseln mit den Basiskomponenten aus SAGA 2.1 oder dem Einer-für-Alle-System (EfA-System), der Begriff, der zukünftig verwendet wird.

die sich allerdings in ihrem Umfang und in ihrer Verfügbarkeit für diverse Basis-Systeme unterscheiden. Ein Beispiel für eine solche Technologie ist die Verwendung von virtuellen Maschinen (VM), in denen Java-Anwendungen laufen. Somit ist eine Java-Anwendung prinzipiell auf jedem System, für das eine VM existiert, lauffähig.

Als Beispiel sei an dieser Stelle auch die Entwicklungs- und Laufzeitumgebung Perl genannt. Wird eine Fachanwendung in der Skriptsprache Perl geschrieben, so ist das Ergebnisprodukt – ähnlich wie in Java – nicht direkt ausführbar. Es wird ein so genannter Perl-Interpreter als Laufzeitumgebung benötigt, in dem das Ergebnisprodukt dann ausgeführt werden kann. Das Ergebnisprodukt kann in der Regel auf jedem Basis-System ausgeführt werden, für welches eine entsprechende Black-Box vorhanden ist. Da es Perl-Interpreter für diverse Basis-Systeme gibt, kann eine Perl-Anwendung grundsätzlich als plattformunabhängig gesehen werden. Was Perl-Interpreter von einem Java-Runtime-System jedoch grundsätzlich unterscheidet, ist der Umfang, der mit dieser Entwicklungs- und Laufzeitumgebung realisierbaren Anwendungen. Als grobe Faustregel kann davon ausgegangen werden, dass größere bzw. komplexere Anwendungen in Java Umgebungen realisiert werden sollten.

Während das Kriterium der Plattformunabhängigkeit von beiden Technologien erfüllt ist, wird die Entscheidung zur Auswahl der Mittel primär durch die funktionalen Eigenschaften einer Entwicklungs- und Laufzeitumgebung und somit eine bessere Gesamtwirtschaftlichkeit bestimmt.

3.1.5 Abhängigkeit aus Sicht des Geschäftsprozesses

Durch die wachsende Verbreitung von Service-Orientierten Architekturen (SOA) wird die Frage der Plattformunabhängigkeit deutlicher als in der Vergangenheit aus Sicht der Umsetzung von Geschäftsprozessen diskutiert. Dies ist grundsätzlich richtig, denn der wesentliche Vorteil einer SOA ergibt sich aus der Entkopplung der Anwendung als konkrete ausführbare Instanz von dem zu erbringenden Service. Dadurch kann aus Sicht des Geschäftsprozesses eine darunter liegende Anwendung ausgetauscht werden, solange der zu erbringende Service soweit standardisiert ist, dass er von einer anderen Anwendung übernommen werden kann.

Daraus ergeben sich zwei Konsequenzen:

1. Die Plattformunabhängigkeit ist keine ausschließliche Eigenschaft von Fachanwendungen, sie kann sowohl „nach unten“ (Basis-System) als auch „nach oben“ (Geschäftsprozess) versetzt werden
2. Ein Geschäftsprozess kann selbst (in sich) plattformunabhängig oder plattformabhängig umgesetzt werden

Die Plattformunabhängigkeit eines Geschäftsprozesses ergibt sich jedoch nicht automatisch aus der gleichen Eigenschaft der verwendeten Anwendungen ge-

nauso, wie in der umgekehrten Richtung. Daraus lässt sich das Verhältnis zwischen der Plattformunabhängigkeit aus Sicht einer Fachanwendung und eines Geschäftsprozesses ableiten:

> *Die Implementierung eines Geschäftsprozesses kann als plattformunabhängig bezeichnet werden, wenn alle (Software)Services dieses Geschäftsprozesses standardisiert sind, in mehreren Anwendungssystemen ausgeführt werden können und somit von einem Austausch dieser profitieren.*

Die Plattformunabhängigkeit aus Sicht des Geschäftsprozesses ersetzt nicht die Plattformunabhängigkeit der Fachanwendungen. Zwar verfolgt sie ein analoges Ziel – der Erhöhung der Wirtschaftlichkeit durch einen möglichen Wechsel der darunter liegenden Schicht – allerdings betrifft ein solcher Wechsel andere Elemente als im Fall der Fachanwendungen. Da jede Ebene über ihre eigenen Randbedingungen der Wirtschaftlichkeit verfügt (z.B. unterliegt der Wechsel von Hardware einem anderen Zyklus als Betriebssystem und dieses einem anderen als eine Fachanwendung), muss die Wirtschaftlichkeit der Plattformunabhängigkeit auf jeder Ebene getrennt betrachtet werden.

3.1.6 Seiten-Abhängigkeiten

Ein ab und zu antreffendes Phänomen ist der Einbau von horizontalen Abhängigkeiten, d.h. Abhängigkeiten, die innerhalb einer Ebene wirksam sind, jedoch nicht über mehrere Ebenen greifen. Ein Beispiel hierfür ist die Bündelung von Produkten im Rahmen einer Lösung oder einer Produktfamilie, zum Beispiel der integrierte Einsatz von Komponenten einer Office-Suite. Eine derartige horizontale Abhängigkeit ist zunächst einmal vernachlässigbar, es sei denn:

- die Abhängigkeiten sind derartiger Natur, dass ein Produkt nicht ohne andere gebündelte Produkte funktionsfähig ist und
- die ergänzenden Produkte des Bundlings in sich plattformabhängig sind

In einem derartigen Fall liegt eine Plattformabhängigkeit vor, die aus der Kopplung mit einer plattformabhängigen Anwendung resultiert.

3.1.7 „Abhängigkeit von der Unabhängigkeit“

Eine wichtige Frage in diesem Kontext ist, ob durch den Einsatz von Programmier- und Laufzeitumgebungen, die zur Entkopplung der Fachanwendung vom darunter liegenden Basis-System nicht neue Plattformabhängigkeiten geschaffen werden. Dies ist sicherlich der Fall, da eine Anwendung grundsätzlich auf ein solches Laufzeitsystem ausgerichtet werden muss und dadurch nicht andere Laufzeitsysteme nutzen kann.

Um dieses Risiko zu minimieren, muss die bei der Gestaltung der Fachanwendung und der Auswahl von Komponenten in jedem Fall auf die Standardisierung geachtet werden.

> *Als Programmier- und Laufzeitumgebungen sollen nur Produkte eingesetzt werden, die standardisiert sind. Richtlinien zum Einsatz bestimmter Standards und Technologien werden in SAGA veröffentlicht.*

Bei nicht standardisierten Produkten, die gegebenenfalls keine breite Industrie-Unterstützung haben, besteht die Gefahr, dass der Aufwand der Plattformunabhängigkeit zu tragen wäre, das Ergebnis jedoch maximal zu einer eingeschränkten Plattformunabhängigkeit führen würde.

Im weiteren Verlauf dieses Kapitels wird auf die Bedeutung der einzelnen Ebenen für die Ausprägung von Plattformunabhängigkeit von Fachanwendungen, näher eingegangen und eine Korrelation mit den Begriffen *Portabilität, Technologieunabhängigkeit, Interoperabilität hergestellt.*

3.1.8 Fazit - Ausprägungen der Plattformunabhängigkeit

Die Plattformunabhängigkeit von Fachanwendungen lässt sich anhand eines Schichtenmodells für konkrete Ebenen in verschiedenen Ausprägungen formulieren, miteinander vergleichen und bewerten. Die Entkopplung von Ebenen durch die Verwendung standardisierter Schnittstellen eines Laufzeitsystems stellt dabei den Maßstab der Plattformunabhängigkeit dar, ohne die positiven Aspekte einer durch die Integration von Komponenten erfolgten positiven Abhängigkeit verneinen zu wollen.

Eine Auswahl der zu erreichenden Plattformunabhängigkeit ist immer spezifisch für die jeweilige Umgebung und wird durch die Wirtschaftlichkeit bestimmt. In diesem Zusammenhang sei auf Kapitel 3.2 verwiesen, in dem verschiedene technische Ansätze vorgestellt werden, mit denen Plattformunabhängigkeit für Fachanwendungen erreicht werden kann. Weitere Informationen zu Wirtschaftlichkeitsbetrachtungen und Kosten-Nutzen Analysen werden in Abschnitt 4.2.2 gegeben.

3.2 Architektur- und Technologie-Ansätze

Mit dem Ausdruck Architektur beschreibt dieser Leitfaden Entscheidungen über den Aufbau von Fachanwendungen. Dabei werden unterschiedliche Parameter einer Fachanwendung wie Plattformunabhängigkeit, Sicherheit oder Leistungsfähigkeit definiert. Ebenfalls Bestandteil einer Architektur ist der Grad der Integration einer Fachanwendung in eine bestehende Systemumgebung.

Dieser Abschnitt stellt Architekturen und Technologien für Lösungsansätze zur Erstellung von plattformunabhängigen Fachanwendungen vor. Ein Schwerpunkt

liegt dabei auf der Erstellung von Fachanwendungen in Eigenregie, unabhängig davon, ob das durch eigenes Personal oder externe Dienstleister erfolgt. Im Hinblick auf den Einsatz von Standardprodukten (COTS[1]) muss die Bewertung der Architektur und der technisch-qualitativen Eigenschaften im Gespräch mit dem Hersteller geklärt werden.

3.2.1 Architekturziele

Es gibt verschiedene Aspekte, die Einfluss auf die Auswahl einer geeigneten Architektur für Fachanwendungen haben.

Natürlich liegt der Sinn einer jeden Architektur darin, die fachlichen und technischen Anforderungen möglichst optimal zu unterstützen. So ist es zum Beispiel fraglich, ob eine Fat-Client Architektur eine Anforderung wie leichte Aktualisierbarkeit der Anwendung erfüllen kann.

Neben der Berücksichtigung solcher fallspezifisch dokumentierten Anforderungen sollte auf jeden Fall auch die bestehende bzw. die zukünftig geplante Systemumgebung betrachtet werden. Da Fachanwendungen in der Praxis nur selten auf der „grünen Wiese“ entstehen, ergeben sich häufig implizite Anforderungen aus der Beschaffenheit der Betriebsumgebung. So kann beispielsweise das Vorhandensein eines Datenbankservers oder einer DOMEA-Registratur durchaus Einfluss auf die Architektur einer Fachanwendung haben.

Unabhängig von konkreten und impliziten Anforderungen gibt es noch eine Reihe von allgemeinen Zielen für Architekturen, wie z.B. Portabilität, Wiederverwendbarkeit und Investitionssicherheit, welche die Forderung Plattformunabhängigkeit unterstützen und weitere Ziele, wie z.B. die Technologieunabhängigkeit, die im Konflikt zur Forderung nach Plattformunabhängigkeit stehen können.

Betrachtet man die Beschreibung von Portabilität in Abschnitt 1.1, dann wird der Zusammenhang zu Plattformunabhängigkeit offensichtlich. Der Zusammenhang zur Wiederverwendbarkeit wird genau dann deutlich, wenn man sich in heterogenen IT-Landschaften bewegt und eine Fachanwendung oder Teile davon auf unterschiedlichen Plattformen, sei es in der eigenen Behörde oder in einer anderen Behörde (z.B. bei EfA-Dienstleistungen), wieder verwenden möchte. Genauso kann bei der Investitionssicherheit die Nachhaltigkeit des Mitteleinsatzes durch Plattformunabhängigkeit dann verbessert werden, wenn schon bei der Beschaffung einer Fachanwendung der Plattformwechsel mittelfristig absehbar ist.

Die Forderung nach Technologieunabhängigkeit kann durchaus im Widerspruch zur Plattformunabhängigkeit stehen, da Plattformunabhängigkeit i.d.R. auf Basis einer bestimmten Technologie (z.B. J2EE) herbeigeführt wird. Andererseits hilft

[1] Commercial Off The Shelf

Plattformunabhängigkeit das Maß an Abhängigkeit zu anderen Technologien zu reduzieren.

Solche Konflikte, die nie ganz auflösbar sein werden, kann es auch zwischen den anderen Architekturzielen, wie z.B. zwischen Technologieunabhängigkeit und Wiederverwendbarkeit geben. So wird z.B. in der Praxis gern auf „Stored Procedures“ zurückgegriffen, die in unterschiedlichen Datenbankanwendungen wieder verwendet werden können. Dies führt allerdings in eine starke Technologieabhängigkeit, die sogar in eine Plattformabhängigkeit führen kann.

Es ist daher eine der wichtigsten Aufgaben beim Entwurf von Architekturen, darauf zu achten, dass die fachlich und technischen Anforderungen und die allgemeinen Architekturziele schlüssig zu einem Gesamtkonzept verbunden werden.

Die vorgestellten Architekturziele sollen vorrangig dabei helfen, bei der Eigenentwicklung von Fachanwendungen einen möglichst offenen softwaretechnischen Entwurf der Anwendung zu erreichen. Weitere Ziele, die dies unterstützen, sind

- Erweiterbarkeit,
- Wartbarkeit,
- Skalierbarkeit und
- Interoperabilität

3.2.2 Anwendungstypen

Es gibt eine Vielzahl von Anwendungstypen, die für den Entwurf von Fachanwendungen eingesetzt werden können. Da es sich bei dem vorliegenden Leitfaden aber nicht um ein Lehrbuch der Softwareentwicklung handelt, ist es nötig, diese große Vielfalt auf eine übersichtliche Menge zu reduzieren. Deshalb werden im Folgenden zwei grundlegende Anwendungstypen vorgestellt, die sich für den Einsatz von plattformunabhängigen Fachanwendungen besonders eignen. Darüber hinaus werden Tipps und Hinweise gegeben, die dabei helfen sollen, mögliche Probleme frühzeitig zu erkennen und zu vermeiden.

Praktisch alle Fachanwendungen lassen sich zwei Anwendungstypen zuordnen. Entweder handelt es sich um eine **Einzelplatzanwendung** oder um eine **Client / Server-Anwendung**.

Eine Einzelplatzanwendungen wird in diesem Leitfaden als Anwendung definiert, in der die gesamte zugrunde liegende Geschäftslogik ebenso wie die Präsentationsebene abgebildet werden. Die Datenhaltung erfolgt lokal auf dem Computer, auf dem die Applikation ausgeführt wird. Das bedeutet, dass eine Einzelplatzanwendung völlig autark von einem Netzwerk betrieben werden kann. Allerdings ist es durchaus möglich, dass Einzelplatzanwendungen mit anderen Anwendungen kommunizieren oder zusätzliche Dienste von ihnen in Anspruch nehmen können.

Ein Beispiel für eine Einzelplatzanwendung ist Microsoft Word. Word kann völlig autark betrieben werden, kann aber über verschiedene Funktionen Dienste andere Anwendungen in Anspruch nehmen und mit dem Internet kommunizieren.

Diese Definition für Einzelplatzanwendungen ist bewusst sehr eng gefasst, um eine klare Trennung zwischen den Anwendungstypen zu erreichen.

Im Gegensatz dazu werden bei Client/Server-Anwendungen die Aufgaben einer Fachanwendung zwischen einem „Dienstleister“ – dem Server – und einem „Nutzer“ – dem Client – aufgeteilt. Üblicherweise verarbeitet bei diesem Modell der Server die von seinen Clients gelieferten Daten zentral nach den Regeln der Geschäftslogik. Darüber hinaus ist er für die Persistenz der Wirkdaten verantwortlich.

In diesem Modell gibt es zwei grundlegende Strategien, wie ein Client aufgebaut sein kann:

- der Thin-Client[1] und
- der Fat-Client[2].

In einer extremen Auslegung des Thin-Client Entwurfs dient der Client als reine Präsentationsebene für die Ein- und Ausgabe der Daten. Die Validierung der Daten erfolgt bei dieser Strategie komplett auf dem Server. Auch werden weder Roh- noch Konfigurationsdaten auf Seite des Thin-Clients abgespeichert.

Etwas anders ist das Vorgehen, wenn ein sehr stark ausgeprägter Fat-Client betrachtet wird. Natürlich übernimmt der Client auch hier wieder die Aufgaben der Präsentationsebene. Zusätzlich werden aber in diesem Fall die Daten – gegebenenfalls zusätzlich zu der Validierung auf dem Server – auf der Clientseite fachlich und technisch vollständig überprüft. Darüber hinaus können sie sogar clientseitig zwischengespeichert werden, um zum Beispiel den Kommunikationsaufwand mit dem Server zu minimieren. Damit handelt es sich bei einem extremen Fat-Client letztendlich um eine Einzelplatzanwendung mit der Möglichkeit zur zentralen Datenspeicherung.

In der Praxis gibt es kaum Beispiele für die jeweilige Extremversion dieser beiden Strategien. Im Normalfall wird ein Thin-Client hauptsächlich einfache technische Überprüfungen der eingegebenen Daten durchführen, wie beispielsweise die Überprüfung von Zahlenformaten, maximalen Textlängen und ähnliches mehr. Auch wird kaum ein Fat-Client die gesamte Geschäftslogik, die bereits auf dem Server vorliegt, erneut implementieren. Vielmehr wird die Logik zwischen Client und Server so aufgeteilt werden, dass dabei nur minimale Überschneidungen

[1] Hier geht es um den Client einer Anwendung (Software), nicht zu verwechseln mit einem Thin-Client der z.B. als PC mit einer minimalen Softwareausstattung innerhalb einer IT-Infrastruktur als Arbeitsplatzrechner eingesetzt wird.

[2] Auch hierbei handelt es sich um den Client einer Anwendung (Software).

anfallen. Unabhängig vom konkreten Client Typen gilt für Client/Server-Anwendungen auf jeden Fall, dass sie ohne ein Netzwerk nicht sinnvoll zu betreiben sind.

Die Unschärfe bei der Abgrenzung der beiden Ausprägungen von Client/Server-Anwendungen ist durchaus beabsichtigt. In der Praxis ist nur schwer ein klarer Punkt zu definieren, an dem nicht mehr von einem Thin-Client, sondern schon über einen Fat-Client gesprochen wird.

An dieser Stelle wird auch klar, warum in diesem Dokument für Einzelplatzanwendungen eine so enge Definition gewählt wurde. Das folgende Beispiel zeigt ein potentielles Abgrenzungsproblem: Eine eigenständige Fachanwendung, in der die komplette Geschäftslogik abgebildet und die Daten lokal gespeichert werden, gilt sicher als Einzelplatzanwendung. Ist diese Einordnung aber auch dann noch richtig, wenn die Daten nicht lokal, sondern auf einem zentralen Datenbankserver gespeichert werden? Mit der sehr engen Definition aus diesem Dokument ist die Antwort ein klares Nein. In diesem Fall wird die Fachanwendung als Client/Server-Anwendung bezeichnet.

Beim Entwurf einer Fachanwendung stellt sich grundsätzlich die Frage, auf Basis welches Anwendungstyps die Applikation entwickelt werden soll. Bei einer solchen Entscheidung kann Abbildung 6 helfen, aus der sich die Schwerpunkte der einzelnen Typen ableiten lassen.

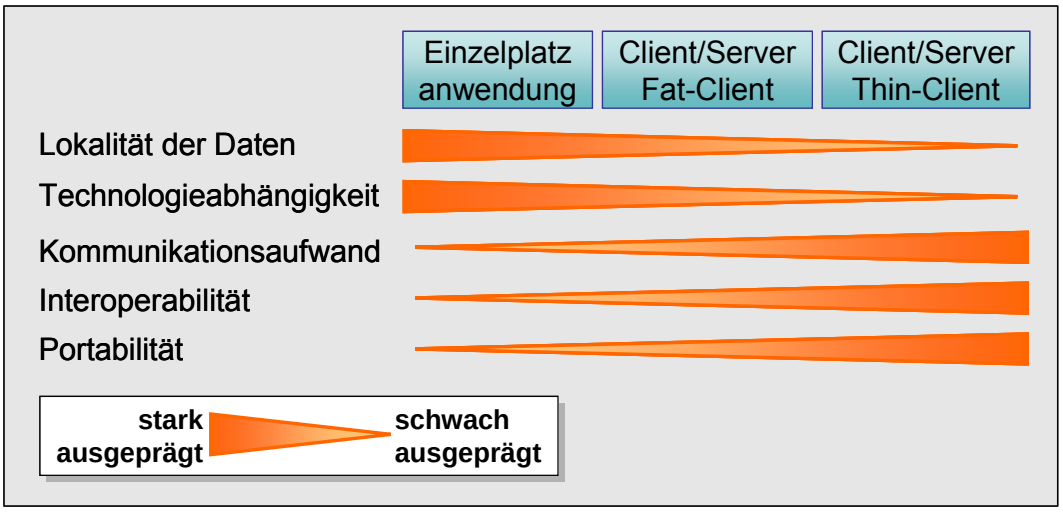


Abbildung 6: Schwerpunkte von Anwendungstypen

Folgendes Beispiel erläutert die Lesart von Abbildung 6: Eine Fachanwendung die ihre Daten – beispielsweise aus Sicherheitsgründen – lokal ablegt, kann optimal als Einzelplatzanwendung umgesetzt werden. Sollen mit der Fachanwendung Daten gesammelt werden, die zu definierten Zeitpunkten an einen Server übertragen werden, ist das die besondere Stärke von Fat-Clients. Dient schließlich eine Fachanwendung nur dazu, Daten zu erfassen, die auf einem zentralen Datenbankserver abgelegt werden, ist der Thin-Client eine gute Wahl.

Neben der Datenhaltung sind die Kommunikationsmöglichkeiten ein weiteres wesentliches Kriterium für die Auswahl des Anwendungstyps. Rahmenbedingungen, in denen kein dauerhafter Zugang zu einem Kommunikationsnetz sichergestellt werden kann, eignen sich nur stark eingeschränkt für Client/Server-Anwendungen. So ist ohne dauerhaften Netzzugang ein Thin-Client überhaupt nicht zu betreiben. Auch bei der Nutzung eines Fat-Client sollte darauf geachtet werden, dass dieser Mechanismen bereitstellt, mit denen er auf den Abbruch oder das Fehlen der Netzverbindung reagieren kann.

Neben rein technischen Überlegungen können auch fachliche Argumente als Entscheidungskriterium für die Auswahl des Anwendungstyps herangezogen werden. Muss eine Fachanwendung gleichzeitig durch viele Benutzer an verschiedenen Lokationen eingesetzt werden, spricht dies eher für eine Client/Server-Applikation. Die Entscheidung, ob ein Thin- oder eher ein Fat-Client implementiert werden soll, kann dabei auch von den fachlichen Rahmenbedingungen abhängen. Thin-Clients können im Normalfall deutlicher leichter als Fat-Clients aktualisiert werden. Daher eignet sich diese Client/Server Ausprägung besonders dann, wenn diese Fachanwendung regelmäßig, zum Beispiel wegen häufig wechselnder Geschäftslogik, aktualisiert werden muss. Darüber hinaus sind sie immer dann sinnvoll, wenn mehrere Personen gemeinsam an einem Datenbestand arbeiten.

Bei der Frage nach der Plattformunabhängigkeit einer Fachanwendung hat der Anwendungstyp hingegen nur eine untergeordnete Bedeutung. Soll die gesamte Fachanwendung plattformunabhängig gestaltet sein, ist das bei beiden Anwendungstypen gleichermaßen leicht bzw. gleichermaßen schwer zu realisieren. Allerdings bieten Client/Server-Anwendungen aufgrund der Zweiteilung der Applikation die Möglichkeit, zumindest plattformunabhängige Clients zu implementieren. Dieses Prinzip kann z.B. durch Web-Applikationen umgesetzt werden, in denen die Benutzerschnittstelle browserbasiert und dadurch plattformunabhängig ist. Allerdings ist es so, dass sich nicht alle Hersteller von Browsern an die Standards halten, und damit browser-basierte Lösung oft nur theoretisch plattformunabhängig sind, da browserspezifische Funktionen von den Anwendungen genutzt werden. Auf der anderen Seite werden Anwendungen häufig so gebaut, dass sie den Browsertyp prüfen und dann die jeweils passende browserspezifische Funktion verwenden. Dies führt jedoch nur zu einer Scheinplattformunabhängigkeit, da neue oder nicht explizit unterstützte Browser ggf. nicht verwendet werden können. Losgelöst von der Problematik der Browser gilt für das Prinzip der Web-Anwendungen: Selbst wenn der Server in Teilen plattformabhängig ist, kann durch die Implementierung einer Fachanwendung als Web-Anwendung eine stark ausgeprägte Plattformunabhängigkeit erreicht werden.

Ein gutes Beispiel sind in diesem Zusammenhang Intranet-basierte Mitarbeiter-Portale. Dabei werden – je nach Art und Ausbaustufe des Portals – Informationen und Prozesse für die Mitarbeiter einer Behörde an einer zentralen Stelle im Intranet der Behörde gemeinsam dargestellt. Technisch liegt dahinter eine

Client/Server-Anwendung, bei der der Server alle nötigen Informationen sammelt und diese dann über eine Internet-Browser basierte Oberfläche, also einem Thin-Client, ausgibt.

Zusammenfassend lassen sich einige Grundsätze definieren, die bei der Auswahl des richtigen Anwendungstyps helfen:

1. Anwendungen, die isoliert lauffähig sein müssen, sollten als Einzelplatzanwendungen realisiert werden.
2. Client/Server-Anwendungen eignen sich optimal für Anwendungen, bei denen Daten zentral gespeichert und validiert werden.
3. Client/Server-Anwendungen, die sich nicht auf eine kontinuierliche Kommunikationsverbindung zwischen Client und Server verlassen können, sollten einen Fat-Client nutzen.
4. Haben Client/Server-Anwendung sehr viele Benutzer, die darüber hinaus im Wesentlichen Aufgaben der reinen Datenerfassung bzw. des Datenabrufs durchführen, eignet sich ein Thin-Client besonders gut.
5. Werden Fachanwendungen aus fachlichen Gründen häufig aktualisiert, lassen sich diese im Normalfall bei einer Client/Server-Anwendung, speziell bei Verwendung eines Thin-Clients, besonders gut durchführen.
6. Werden verschiedene einzelne Anwendungen unter einer gemeinsamen Oberfläche verbunden, eignet sich der Client/Server Ansatz gut. Meist wird die gemeinsame Oberfläche dabei als Thin-Client realisiert. Häufig spricht man in diesem Zusammenhang auch von „Portalen“.

3.2.3 Architekturmuster

Die Situation bei Softwarearchitekturen ähnelt stark der von Anwendungstypen. Auch hier gibt es eine große Vielfalt an unterschiedlichsten Mustern, auf deren Basis Fachanwendungen realisiert werden können. Um die Übersichtlichkeit zu wahren, konzentriert sich dieser Leitfaden daher im Weiteren auf folgende vier Architekturmuster:

- Monolithen
- n-Schichten Architekturen
- Service orientierte Architekturen
- Komponentenbasierte Architekturen

3.2.3.1 Monolithen

Das erste dieser vier Architekturmuster ist schnell zu erklären. Wie der Name bereits suggeriert, handelt es sich bei einer monolithischen Architektur um einen ganzheitlichen Ansatz. Diese Softwarearchitektur sieht keinerlei Gliederung einer

Fachanwendung vor, weder nach fachlichen noch technischen Aspekten. Streng genommen sollte im Zusammenhang mit Monolithen nicht von Architekturen gesprochen werden, da keines der in Abschnitt 3.2.1 genannten Architekturziele planmäßig verfolgt wird.

Dennoch finden sich Monolithen relativ häufig in modernen IT-Infrastrukturen. So basieren zum Beispiel viele kleine Werkzeuge auf einer solchen Architektur. Der Grund dafür ist einfach. Nicht selten werden diese Werkzeuge von einzelnen Mitarbeitern in einer Art „Guerillataktik“ entwickelt, um sie bei der Bearbeitung eines sehr speziellen Problems ihres Aufgabenbereichs zu unterstützen. Dazu werden häufig Makro- oder Skriptsprachen von Standardanwendungen eingesetzt. Bei solchen Entwicklungen wird im Normalfall ausschließlich der funktionale Nutzen der entstehenden Anwendung gesehen. Daher sind die resultierenden Anwendungen wegen der monolithischen Architektur zwar technisch problematisch, häufig aber funktional äußerst hochwertig.

Problematisch wird eine monolithische Softwarearchitektur vor allem dann, wenn die darauf basierenden Anwendungen erweitert werden sollen. Diese Erweiterungen sind dann nicht selten nur durch den ursprünglichen Entwickler und auch nur in begrenztem Umfang durchführbar.

Gerade die Schwächen bei der Erweiterbarkeit und Wartbarkeit machen Monolithen trotz ihres durchaus vorhandenen fachlichen Nutzens zu problematischen Elementen einer IT-Infrastruktur. Handelt es sich bei den Monolithen letztendlich um Makros für Standardanwendungen, entsteht zudem noch eine Technologieabhängigkeit, durch welche die Situation zusätzlich verschärft wird. Ziele wie Plattformunabhängigkeit werden letztlich nur durch Zufall erreicht. Zum Beispiel dann, wenn die Standardanwendung, für die die Makros entwickelt wurde, plattformunabhängig ist oder für die Entwicklung eine plattformunabhängige Programmiersprache gewählt wurde. Aus diesem Grund werden monolithische Softwarearchitekturen in der professionellen Softwareentwicklung praktisch nicht eingesetzt.

3.2.3.2 n-Schichten-Architekturen

Einen vollständig anderen Ansatz verfolgen Architekturen, die mit eindeutig voneinander abgegrenzten Abstraktionsschichten arbeiten. Grob beschrieben ist bei einer solchen n-Schichten Architektur jede der „n“ Schichten für einen Aufgabenbereich der Applikation verantwortlich. Damit wird die Funktionalität einer jeden Schicht von den anderen Schichten abgekapselt. Die Kommunikation und der Datenaustausch zwischen den Schichten erfolgt über klar definierte Schnittstellen und im Idealfall nur zwischen benachbarten Schichten.

Der prominenteste Vertreter dieses Architekturmusters ist die Drei-Schichten Architektur, die zum Beispiel bei Client/Server-Anwendungen häufig verwendet wird. Bei diesem Ansatz wird eine Fachanwendung wie in Abbildung 7 dargestellt, in folgende drei Schichten zerlegt:

- Präsentationsschicht
- Logikschicht
- Datenhaltungsschicht

Die oberste Schicht bei der Drei-Schichten Architektur ist die **Präsentationsschicht**. Diese enthält die Benutzerschnittstelle und ist damit für die Verarbeitung der Benutzereingaben und die Aufbereitung der Systemausgaben verantwortlich. Eine fachliche Validierung der erfassten Daten wird in der Präsentationsschicht nicht durchgeführt, allerdings werden die Daten i.d.R. auf technische Korrektheit überprüft.

Die **Logikschicht** ist die mittlere Schicht in diesem Modell. Sie verantwortet die Steuerung der Anwendung und implementiert die zugrunde liegende Geschäftslogik. Dazu werden in dieser Schicht Entscheidungen und Berechnungen auf Basis der Geschäftslogik durchgeführt, Daten fachlich validiert und Prozesse angestoßen. Ebenso ist es Aufgabe der Logikschicht, die Datenkommunikation zwischen den beiden sie umgebenden Schichten zu steuern und gegebenenfalls die Daten für die jeweilige Schicht aufzubereiten.

Auf unterster Ebene ist in der Drei-Schichten Architektur die **Datenhaltungsschicht** angesiedelt. Wie der Name bereits andeutet, liegt die Aufgabe dieser Schicht in der Verwaltung der anfallenden Daten. Dabei ist es unwichtig, wie die Daten konkret abgelegt werden, ob im Dateisystem oder in einer Datenbank. Wichtig ist, dass die Daten in dieser Schicht nur abgelegt bzw. geladen werden. Die Verarbeitung der Daten erfolgt exklusiv in der Logikschicht, ihre Visualisierung in der Präsentationsschicht. Im Normalfall gibt es keine direkte Verbindung zwischen der Datenhaltungsschicht und der Präsentationsschicht.

Abbildung 7: Schwerpunkte von Anwendungstypen

Diese funktional geschnittenen Schichten der Drei-Schichten Architektur finden sich prinzipiell in jeder n-Schichten Architektur wieder. Zusätzliche Schichten können durch eine Verfeinerung der Logikschicht entstehen. So basieren zum Beispiel viele Webapplikationen auf einer Vier-Schichten Architektur. Dabei werden die Funktionen, die für die Sicherheit der Anwendung verantwortlich sind, aus der Logikschicht gelöst und in eine eigene Schicht zwischen Präsentationsschicht und Logikschicht ausgelagert.

Die n-Schichten Architekturen bieten eine Reihe von Vorteilen. So erfüllen sie die in Abschnitt 3.2.1 vorgestellten Architekturziele Erweiterbarkeit, Wartbarkeit und Portabilität besonders gut, da die einzelnen Schichten wie autarke Module gesehen werden können. Als solche können sie getrennt bearbeitet und getestet werden. Wurde beim Entwurf der Schichten darauf geachtet, dass ihre Schnittstellen sauber definiert sind, lassen sich die einzelnen Schichten sogar durch andere austauschen. Eine solche Austauschbarkeit kann für das Erreichen von Plattformunabhängigkeit äußerst nützlich sein. Dabei ist es unerheblich, ob es sich bei der auf einer n-Schichten Architektur basierenden Anwendung um eine Einzelplatzanwendung oder um ein Client/Server-Applikation handelt.

Speziell zu den Drei-Schichten Architekturen ist anzumerken, dass sie sich gut als Basis für Client/Server-Anwendungen eignen. Da die einzelnen Schichten über klar definierte Schnittstellen kommunizieren, ermöglicht es dieser Ansatz, eine hohe Plattformunabhängigkeit zu erreichen. Beispielhaft dafür seien Web-

anwendungen erwähnt. Während Datenhaltung, Logik und in der Regel auch die Präsentationsschicht auf einem Server des Betreibers gehalten werden, über nimmt der Web-Browser des jeweiligen Benutzers die Visualisierung der Oberfläche. Ordnet man den Client einer eigenen Schicht zu, spricht man von einer Vier-Schicht-Architektur[1].

Wie so häufig gilt auch bei n-Schichten Architekturen: Wo es Licht gibt, gibt es auch Schatten. So führt die Kommunikation durch die Schichten zu Effizienzverlusten, da Daten und Meldungen immer über verschiedene Schichten transportiert werden müssen. Darüber hinaus kann es durchaus Probleme bereiten, eine klare und saubere Abgrenzung der Schichten zu definieren.

Trotz dieser Probleme gehören die n-Schichten Architekturen zu den am weitesten verbreiteten Architekturen in der Softwaretechnik.

3.2.3.3 Service orientierte Architekturen (SOA)

Bei der dritten in diesem Leitfaden näher erläuterten Architektur handelt es sich um serviceorientierte Architekturen, die häufig auch nur mit dem Kürzel SOA bezeichnet werden.

Hinter SOA steht die Idee, dass fachliche Funktionalitäten in Form von Diensten, so genannten Services, dezentral bereitgestellt werden. Diese Services lassen sich dann über standardisierte Schnittstellen in Fachanwendungen bzw. weitere Services integrieren.

Statt also die Welt für jede Fachanwendung neu zu erfinden, werden bei der Implementierung des Systems im wesentlichen Aufrufe von wiederverwendbaren, voneinander unabhängigen und lose gekoppelten Services aneinander gereiht. Anders als bei der Schichtenarchitektur findet sich damit die Geschäftslogik nicht mehr in einer einzelnen Schicht oder in einem einzelnen Programm, sondern verteilt sich über mehrere unabhängige Dienste. Letztlich kann eine Fachanwendung, die auf Basis einer SOA aufgebaut wurde, als eine Art „virtuelles System“ bezeichnet werden.

Häufig werden als technische Voraussetzungen von SOA verschiedene Protokolle und Techniken wie Web Services, SOAP oder WSDL genannt. Das ist nur begrenzt richtig, da SOA ein wirklich unabhängiges Architektur-Paradigma ist, das auf jeder dienstbasierten Technologie aufgebaut werden kann.

Eine Besonderheit von serviceorientierten Architekturen ist, dass die Services unabhängig von zugrunde liegenden Implementierungen über Schnittstellen aufgerufen werden. Die Spezifikation dieser Schnittstellen ist dabei öffentlich zugän-

[1] Siehe Kapitel 6 “Computational Viewpoint: Referenz-Software-Architektur“ in SAGA 2.1 (www.kbst.bund.de/saga)

gig. Abbildung 8 zeigt schematisch den daraus resultierenden Aufruf eines Services. Dabei sucht der Service Consumer als Nutzer beim Service Broker nach dem benötigten Dienst. Falls ein entsprechender Dienst durch den Anbieter, den Service Provider, beim Service Broker registriert wurde, stellt dieser dem Consumer die nötigen Daten für eine Verbindung bereit. Mit Hilfe dieser Daten kann dann der Consumer den Dienst vom Provider in Anspruch nehmen.

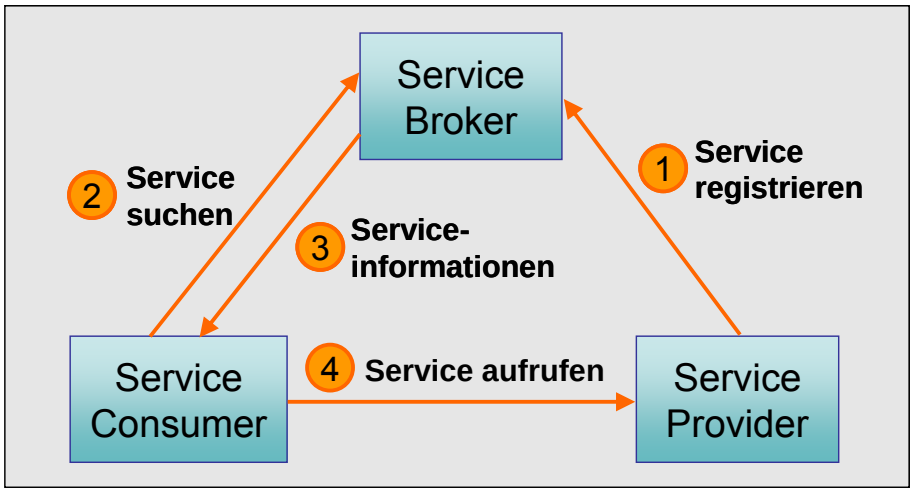


Abbildung 8: Schematischer Aufruf eines Services in einer SOA

Da Services nur auf Grundlage der öffentlich zugängigen Schnittstelle kommunizieren, können sie in unterschiedlichen Programmiersprachen und auf unterschiedlichen Systemplattformen realisiert werden. Daher eigenen sich SOA sehr gut für die Entwicklung von plattformunabhängigen Fachanwendungen. Neben dieser Technologieunabhängigkeit und Portabilität sind die Architekturziele Wiederverwendbarkeit und Interoperabilität Schwerpunkte von serviceorientierten Architekturen. Auch bei diesen Zielen hilft, dass SOA nicht an bestimmte technische Vorgaben gebunden sind, sondern in unterschiedlichen Umgebungen implementiert werden können. Folglich werden SOA häufig zur Anwendungsintegration genutzt und helfen damit auch, ein hohes Maß an Investitionssicherheit für die bestehende IT-Infrastruktur zu erreichen.

Allerdings wird für die Serviceinteraktion eine Kommunikationsinfrastruktur vorausgesetzt. Das bedeutet, dass dieses Architekturmuster nicht beim Entwurf von Einzelplatzanwendungen sondern nur für Client/Server-Applikationen eingesetzt werden kann. Ein weiterer Nachteil von serviceorientierten Architekturen ist, dass die anfänglichen Aufwendungen bei der Einführung von SOA erheblich sein können. So ist es i.d.R. nötig, die Geschäftslogik, die in bereits vorhandenen Fachanwendungen abgebildet ist, mit Hilfe von Adaptern zu Services zu erweitern.

3.2.3.4 Komponentenbasierte Architekturen

Der vierte und letzte Architekturtyp, der in diesem Leitfaden betrachtet wird, ist die komponentenbasierte Architektur. Eine der grundlegenden Ideen dieser Architektur ist der Gedanke, dass sich Fachanwendungen durch die Koppelung von bereits erstellten Bausteinen, oder um in der Terminologie zu bleiben, Komponenten, zusammensetzten lassen. Obwohl SOAs und komponentenbasierte Architekturen damit der gleichen Grundidee folgen, unterscheiden sie sich in ihrer technischen Realisierung deutlich voneinander.

Kernbestandteil einer jeden komponentenbasierte Architektur ist ein zentrales Element, eine so genannte Middleware. Sie verbindet alle Komponenten einer Fachanwendung miteinander und ist für die Kommunikation und den Datenaustausch zwischen ihnen verantwortlich (siehe auch Abschnitt 4.1.3.5).

Im Vergleich dazu ist bei SOAs eine solche zentrale Komponente zwar möglich aber nicht nötig. Auch sind die Komponenten einer komponentenbasierten Architektur im Normalfall deutlich enger miteinander verbunden, als das die Services einer SOA sind.

Prinzipiell gibt es keine Standards oder Vorgaben, denen Middleware-Produkte folgen müssen. Allerdings basieren viele dieser Produkte auf den Plattformen .NET von Microsoft, J2EE von Sun oder CORBA von der OMG.

Der große Vorteil von komponentenbasierten Architekturen ist, dass diese mit klar definierten Schnittstellen arbeiten. Somit lassen sich einzelne Komponenten im Normalfall problemlos durch andere mit gleichem Funktionsinhalt austauschen.

3.2.3.5 Fazit zu Architekturmuster

Zusammenfassend zeigt sich in diesem Kapitel, dass Architekturen die Plattformunabhängigkeit von Fachanwendungen zwar unterstützen, niemals aber jedoch garantieren oder verhindern können. So kann ein Monolith, der in einer plattformunabhängigen Programmierplattform wie Java entwickelt wurde, in höchstem Maß plattformunabhängig sein. Im Gegensatz dazu ist selbst eine Fachanwendung, der eine serviceorientierten Architektur zugrunde liegt, nicht davor geschützt, plattformabhängig zu sein, wenn sie beispielsweise auf spezielle Betriebssystemdienste zugreift.

3.2.4 Standards und Schnittstellen

Offene, allgemein anerkannte Standards und Schnittstellen sind von zentraler Bedeutung für plattformunabhängige Fachanwendungen. Die schönsten Architekturen und die besten Techniken helfen nicht weiter, wenn eine Fachanwendung – aus welchem Grund auch immer – auf proprietären Standards oder

Schnittstellen aufsetzt. Daher müssen offene Standards und offene Schnittstellen integraler Bestandteil einer jeden plattformunabhängigen Fachanwendung sein.

Dabei wird eine Schnittstelle unabhängig von der eingesetzten Technologie dann als „offen“ bezeichnet, wenn die auszutauschenden Daten sowie die Schnittstelle selbst ausführlich und detailliert dokumentiert sind und diese Dokumentation jedem frei und kostenlos zur Verfügung gestellt wird.

In der heutigen Zeit, in der der Austausch von Daten, Informationen und Wissen zwischen verschiedenen Personen, Unternehmen und Behörden eine immer größere Bedeutung gewinnt, spielen offene Standards eine zunehmend wichtigere Rolle.

Durch offene Standards entsteht in der Regel ein breiteres Spektrum von unterschiedlichen, konkurrierenden Lösungen. Auf diesem Wege steht den Endkunden meist eine größere Auswahl von Lösungsalternativen zur Verfügung, wodurch prinzipiell eine mögliche Abhängigkeit reduziert wird.

Zwischen Standards und den Schnittstellen von Fachanwendungen existiert ein enger thematischer Zusammenhang. Schnittstellen sollten immer Standards genügen. Dementsprechend sollten sich Schnittstellen von plattformunabhängigen Fachanwendungen an offenen Standards orientieren. Im deutschen eGovernment werden z.B. fachspezifische XML-basierte Standards für den behördenübergreifenden Datenaustausch für die öffentliche Verwaltung (XÖV) im Rahmen von Deutschland online definiert (z.B. XMeld, XBau , XJustiz, XSozial). Diese sind in der Verwaltung anerkannt, dauerhaft offen gelegt und stehen sowohl der Verwaltung als auch den Softwareherstellern dauerhaft frei zur Verfügung.

Durch offene, standardbasierte Schnittstellen können Komponenten unterschiedlicher Technologien, z. B. CORBA, J2EE oder Microsoft .NET, bzw. Produkte unterschiedlicher Hersteller innerhalb einer IT-Infrastruktur eingesetzt werden. Die daraus resultierende Unabhängigkeit sichert eine Softwarevielfalt und ermöglicht den Nutzern, sich frei für die aus ihrer Sicht beste Lösung zu entscheiden.

3.2.5 Technologien und Techniken

Neben der eher abstrakten Betrachtung von Anwendungstypen und Architekturen gibt es eine Reihe konkreter Technologien und Techniken, die dabei helfen, plattformunabhängige Fachanwendungen zu erhalten.

Die Techniken sollen im Rahmen dieses Leitfadens jedoch nur grob skizziert werden. Darüber hinaus ist auf den Band „Standards und Architekturen für E-Government-Anwendungen“ (SAGA) innerhalb der Schriftenreihe der KBSt zu verweisen. Dieses Dokument, welches sich intensiv mit konkreten Standards und Architekturen für die Entwicklung von E-Government-Anwendungen befasst, wird durch die KBSt kontinuierlich fortgeschrieben. An dieser Stelle sei die Lektüre von SAGA mindestens für all jene dringend empfohlen, die sich in ihrem beruflichen Alltag mit der Erstellung und Beschaffung von Fachanwendungen für den

öffentlichen Sektor befassen. Für den weiteren Verlauf des vorliegenden Dokumentes gilt: Wann immer auf SAGA verwiesen wird, ist damit die zur Zeit der Erstellung des Leitfadens gültige Version 3.0 gemeint.

Im Verlauf dieses Abschnitts werden verschiedene aktuelle Standards, Technologien und Techniken kurz vorgestellt, durch die Plattformunabhängigkeit bei Fachanwendungen gefördert wird. Dabei werden die einzelnen Techniken zwar genannt, nicht aber ausführlich erklärt. Damit wird für das Verständnis dieses Kapitels ein fundiertes (Software-)technisches Basiswissen seitens des Lesers vorausgesetzt.

3.2.5.1 Programmiersprachen

Eine der wichtigsten Entscheidungen, um plattformunabhängige Fachanwendungen zu erhalten, ist die Wahl der zugrunde liegenden Programmiersprache. Obwohl sich mit Programmiersprachen wie C und C++ unter Einsatz diverser Hilfsmittel in Bezug auf Plattformunabhängigkeit durchaus respektable Ergebnisse erzielen lassen, empfiehlt dieser Leitfaden den Einsatz von Java. Dabei gilt die Faustregel: Für die Entwicklung von Einzelplatzanwendungen sollte auf Java 2 Standard Edition zurückgegriffen werden. Sollen allerdings Client/Server-Applikationen entwickelt werden, eignet sich die Java 2 Enterprise Edition deutlich besser. Der Grund hierfür ist, dass Java für eine Vielzahl von Betriebssystemen erhältlich ist und nicht wie .NET derzeit nur für Microsoft Windows in einer ausgereiften Implementierung zur Verfügung steht[1]. Anwendungen, die in Java entwickelt wurden, sind im Normalfall Bytecode-kompatibel. Das bedeutet, dass ein und dieselbe Version der Anwendung auf allen Betriebssystemen läuft, die die zugrunde liegende Java-Version unterstützen. Gemäß SAGA ist der Einsatz von Java als obligatorisch[2] kategorisiert.

Allerdings gibt es auch hierbei eine Ausnahme: Setzt die bestehende IT-Infrastruktur massiv auf den Einsatz von Microsoft Windows Betriebssystemen, kann es interessant sein, das Microsoft .NET Framework näher zu betrachten. Die Konzeption dieses Frameworks ist prinzipiell darauf ausgelegt plattformunabhängige Fachanwendungen zu entwickeln und zu nutzen, allerdings existiert, wie bereits im vorangegangenen Absatz beschrieben, derzeit nur für Microsoft Windows eine ausgereifte Implementierung des Frameworks. Eine allgemeine Implementierung namens „Mono“, die unter anderem auch Linux und Unix Systeme anbindet, ist aber bereits in Entwicklung und wird auch schon für die Entwicklung

1 siehe auch nachfolgenden Absatz

2 Zur Bedeutung der Klassifizierung dieser Technologien siehe Kapitel „Klassifizierung und Lebenszyklen von Standards“ in SAGA 3.0 (www.kbst.bund.de)

und den Betrieb von Anwendungen eingesetzt[1]. Darüber hinaus steht .NET in SAGA „unter Beobachtung“ [2]

3.2.5.2 Schnittstellen

Neben der Programmiersprache haben Schnittstellen für die Plattformunabhängigkeit eine enorme Bedeutung. Das gilt umso mehr, je stärker die einzelnen Fachanwendungen mit anderen Fachanwendungen oder der IT-Infrastruktur integriert werden müssen. Vor allem sind dabei folgende drei Schnittstellentypen zu betrachten:

- Kommunikationsschnittstellen Client zu Server und Server zu Server
- Kommunikations-Applikationen im Allgemeinen
- Anbindung von Applikationen

Für die Anbindung von Applikationen gibt es im Wesentlichen drei Standards, deren Verwendung in plattformunabhängigen Fachanwendungen unabdingbar ist. Sehr viele Fachanwendungen speichern ihre Daten in Datenbanksystemen. Für die dabei benötigte Anbindung der Datenbanken an die Fachanwendungen gibt es den ausgereiften Standard JDBC. Allerdings setzt der Einsatz von JDBC zum einen voraus, dass als Programmiersprache Java gewählt wurde, zum anderen muss das Datenbanksystem eine entsprechende Schnittstelle zur Verfügung stellt. JDBC ist in SAGA als obligatorisch[3] gekennzeichnet.

Als Syntax für den Austausch von Daten zwischen zwei beliebigen Anwendungen eignet sich besonders gut die Extensible Markup Language oder kurz XML und die dazugehörige Web-Service Schnittstellendefinition, die mit der Web-Service Definition Language (WSDL) definiert ist. Bei der Erstellung oder der Beschaffung von Fachanwendungen sollte darauf geachtet werden, dass diese eine WSDL- Schnittstelle sowie eine XML-Definition für den Austausch von Daten bereitstellen. In der aktuellen Version von SAGA werden XML und WSDL als obligatorisch eingestuft[4].

---

1 Der Einsatz von Mono kann in sofern problematisch werden, dass Microsoft die Patentrechte auf .NET besitzt und für die Zukunft nicht sichergestellt ist, dass Microsoft keine Ansprüche geltend macht.

2 Zur Bedeutung der Klassifizierung dieser Technologien siehe Kapitel „Klassifizierung und Lebenszyklen von Standards“ in SAGA 3.0 (www.kbst.bund.de)

3 Zur Bedeutung der Klassifizierung dieser Technologien siehe Kapitel „Klassifizierung und Lebenszyklen von Standards“ in SAGA 3.0 (www.kbst.bund.de)

4 Zur Bedeutung der Klassifizierung dieser Technologien siehe Kapitel „Klassifizierung und Lebenszyklen von Standards“ in SAGA 3.0 (www.kbst.bund.de) sowie zur Klassifizierung XML Kapitel 8.2.3 und WSDL Kapitel 8.6.1.1.

Ein weiterer, in SAGA empfohlener Standard für die Programm-Programm-Kommunikation sind Web Services. Dieser Standard ist vor allem bei der Realisierung von serviceorientierten Architekturen von besonderer Bedeutung[1].

Neben der Anbindung von Anwendungen gibt es auch eine Reihe von Fachanwendungen, die Kommunikationsaufgaben wie das Senden von Emails, das Aufrufen von Webseiten u. ä. zu erledigen haben. Bei diese Aufgaben ist darauf zu achten, dass die Fachanwendungen dafür anerkannte Protokolle wie beispielsweise das Simple Mail Transfer Protokoll (SMTP) für das Versenden von Emails unterstützen. Auch hier gibt SAGA konkrete Hinweise zu den als obligatorisch[2] eingestuften Protokollen.

Für Client/Server-Applikationen haben Kommunikationsschnittstellen vom Client zum Server aber auch von Server zu Server ein große Bedeutung. Speziell sollte darauf geachtet werden, dass die Spezifikation der zu übertragenden Daten entweder in der XML Schema Definition (XSD) oder dem Standard Relax NG erfolgt. Darüber hinaus hilft das Simple Object Access Protocol (SOAP) dabei, Daten als strukturierte Objekte zwischen Anwendungen auszutauschen.

Letztendlich lassen sich alle der oben aufgeführten Hinweise für die Schnittstellen von plattformunabhängigen Fachanwendungen wie folgt zusammenfassen: Schnittstellen müssen offenen Standards genügen (siehe auch Abschnitt 3.2.4).

Für die zu übertragenden Daten gilt darüber hinaus, dass XML mit entsprechenden Schemadefinitionen eine hervorragende Grundlage ist. Dies gilt nicht nur für den öffentlichen Sektor, jedoch besonders hierfür. So gibt es z.B. die für den Einsatz in Städten und Kommunen entwickelten Online Services Computer Interface (OSCI) Standards. Hierzu zählen u.a. die bereits weiter oben erwähnten XÖV-Standards. Zur Pflege dieser wurde die OSCI-Leitstelle eingerichtet. Näheres hierzu findet man unter http://www1.osci.de/.

3.2.5.3 Architekturen

Ein letzter Aspekt, durch den Plattformunabhängigkeit von Fachanwendungen zumindest in gewissem Maß unterstützt werden kann, ist die Auswahl einer geeigneten Architektur. Wie bereits in Abschnitt 3.2.3 erwähnt, gibt es nicht die eine Architektur, die Plattformunabhängigkeit garantiert. Allerdings haben sich n-Schichten-Architekturen sowohl für Einzelplatzanwendungen als auch bei Client/Server-Applikationen bewährt. Die Verwendung dieses Architekturmusters bietet die Möglichkeit, wenn auch nicht die gesamte Fachanwendung dann doch

1 Eine umfassende Beschreibung von Web-Services im Kontext von eGovernment wurde von der BITKOM Arbeitsgruppe eGovernment erstellt – „Leitfaden Web-Services“; Version 2.0, Oktober 2005

2 Zur Bedeutung der Klassifizierung dieser Technologien siehe Kapitel „Klassifizierung und Lebenszyklen von Standards“ in SAGA 3.0 (www.kbst.bund.de)

zumindest einzelne Teile plattformunabhängig zu gestalten. Dies gilt vor allem, wenn es sich bei der Fachanwendung um eine Client/Server-Applikation handelt. Um dabei optimale Plattformunabhängigkeit zu erhalten, drängt sich die Entwicklung von Webapplikationen nahezu auf.

Abschließend einige Anmerkungen zu serviceorientierten Architekturen: Sie entwickeln sich immer stärker zu einer Ergänzung zu den n-Schichten-Architekturen, die Konzepte und erforderliche Standards für den Austausch von Information zwischen den Komponenten einer Komponentenarchitektur über offene Dienstschnittstellen, wie Web-Services, definieren. Allerdings ist das Thema SOA in seiner aktuellen Ausprägung noch relativ neu. SOA scheint die Lösung für viele Probleme – unter anderem auch Plattformunabhängigkeit – der Softwareentwicklung zu sein, die auch Lösungen für unterschiedliche Aspekte der Plattformunabhängigkeit anbietet. Hier ist der Leser gefordert, die Entwicklung von SOA weiter zu verfolgen, um sich eine eigene Meinung bilden zu können.

3.2.6 Fazit zu Architektur- und Technologieansätzen

Ein Ergebnis von Abschnitt 3.2 ist, dass es nicht den einen Weg zum Ziel Plattformunabhängigkeit gibt. Daraus resultiert unmittelbar der Vorteil, dass es mehrere Ansätze gibt, um plattformunabhängige Fachanwendungen zu erhalten. Dies bedeutet allerdings auch, dass die Personen, die sich für die Beschaffung oder Entwicklung von Fachanwendungen verantwortlich zeigen, verschiedene Aspekte gegeneinander abwiegen müssen.

Die nachfolgende Tabelle 2 fasst noch einmal die wichtigsten Aussagen aus dem Abschnitt 3.2 in Übersichtsform zusammen. Die Tabelle stellt eine Übersicht dar mit deren Hilfe eine erste grobe Eingrenzung bei der Wahl von Anwendungstypen, Architekturmustern und Technologien zur Gestaltung oder Auswahl einer plattformunabhängigen Fachanwendung erfolgen kann. Sie ersetzt aber in keinem Fall eine konkrete Bewertung der jeweils aktuellen Situation. Zur Wahl der richtigen Architektur und Technologie bezogen auf die jeweilige Situation liefert Kapitel 4 die notwendigen Hinweise.

Wesentliche Hilfsmittel bei der Beschaffung oder Entwicklung von plattformunabhängigen Fachanwendungen sind vor allem ein gehöriges Maß an Erfahrung und der gesunde Sachverstand.

Die nachfolgende Tabelle 2 fasst die Empfehlungen aus Kapitel 3.2 zusammen, welche durch die Klassifizierungen in SAGA unterstützt werden.

Tabelle 2: Zusammenfassung der Empfehlungen aus Kapitel 3.2

Thema	Empfehlungen
Anwendungstyp	Die Wahl für eine Einzelplatz- oder eine C/S-Anwendung ist stark abhängig von den jeweiligen fachlichen Anforderungen. Wenn möglich sollte der Typ einer Client/Server-Anwendung und hier vor allem die Webanwendung oder das Portal gewählt werden.
Architekturmuster	Gut geeignet sind n-Schichten- und Service orientierte Architekturen. Komponentenbasierte Architekturen können ebenfalls genutzt werden, besitzen aber die Einschränkung, dass i.d.R. die Bindung zwischen den Komponenten höher ist. Monolithen sind mit Blick auf die Plattformunabhängigkeit eher weniger geeignet.
Programmiersprache	Hier ist Java zu empfehlen, da Java für viele Betriebssysteme verfügbar ist. Dies gilt nicht für .NET, das in einer vollständigen Implementierung nur für Windows verfügbar ist.
Schnittstellen	Schnittstellen sind immer dann gut geeignet, wenn sie offenen Standards folgen. Die Erfahrungen haben gezeigt, dass insbesondere XML-basierte Schnittstellen und Protokolle sich gut eigenen, um Unabhängigkeit zu erreichen.

Darüber hinaus lässt sich aus den bisher betrachteten Zusammenhängen eine Frageliste ableiten, mit deren Beantwortung die Ausprägung der Plattformunabhängigkeit einer bestehenden Fachanwendung auf einem recht hohen Abstraktionsniveau geprüft werden kann.

Frageliste

Aufgabe: Festellen der Ausprägung von Plattformunabhängigkeit einer Fachanwendung.

1. Verwendet die Fachanwendung ein Entwicklungs- und Laufzeitsystem zur Entkopplung vom Basis-System?
2. Ist das durch die Fachanwendung verwendete Laufzeitsystem auf mehreren Basis-Systemen lauffähig?
3. Kann die Fachanwendung unabhängig von einer bestimmten Version der Laufzeitumgebung genutzt werden?
4. Folgen die Schnittstellen zur Anbindung der Fachanwendung an andere Systeme anerkannten Standards?
5. Sind die Schnittstellen öffentlich dokumentiert?

6. Sind die entsprechenden APIs offen gelegt und dokumentiert?
7. Ist das Laufzeitsystem weitgehend unabhängig vom Basis-System?
    a. Wird bestimmte Hardware benötigt bzw. darf bestimmte Hardware nicht eingesetzt werden?
    b. Wird Infrastruktursoftware, beispielsweise Datenbank-Management-Software, benötigt?
    c. Müssen spezielle Produkte eines bestimmten Herstellers genutzt werden?
    d. Muss diese Software in bestimmten Versionen vorliegen?
8. Falls die Fachanwendung an andere Anwendungen gebunden ist, ist die Bindung (Nutzung) so gestaltet, dass die Fachanwendung auch ohne diese anderen Anwendungen sinnvoll genutzt werden kann?

Immer wenn die Hauptfragen (1 bis 9) mit nein beantwortet werden, besteht eine mehr oder weniger hohe Gefährdung der Plattformunabhängigkeit. Je mehr der Hauptfragen mit nein beantwortet werden, um so geringer ist die Ausprägung der Plattformunabhängigkeit. Handelt es sich um eine bestehende Fachanwendung, dann sollte kurz- bis mittelfristig Abhilfe geschaffen werden. Entweder an den betroffenen Stellen (Antwort = nein) oder durch Austausch der Fachanwendung. Worauf hierbei zu achten ist, wird in den vorangegangen und nachfolgenden Kapiteln beschrieben.

3.3 Integration und Plattformunabhängigkeit

In den vergangenen Kapiteln wurde das Thema Plattformunabhängigkeit überwiegend isoliert für Fachanwendungen betrachtet. Allerdings existieren Fachanwendungen nicht im luftleeren Raum, sondern sind Bestandteil der gesamten IT-Infrastruktur einer Behörde. Dabei können die Beziehungen zwischen diesen – teilweise in Art und Komplexität sehr unterschiedlichen – Bestandteilen von friedlicher Koexistenz bis zu enger Abhängigkeit reichen.

Nach den langen Jahren des – zeitweise sogar unkontrollierten – Wachstums von IT-Infrastrukturen ist es so, dass gerade im öffentlichen Sektor, ein Wechsel in den Schwerpunkten der IT-Strategie bevorsteht. Themen wie die Konsolidierung und Integration von IT-Infrastrukturen gewinnen immer mehr an Bedeutung und sollen daher auch in diesem Dokument nicht unberücksichtigt bleiben.

Die Integration von Anwendungen bedeutet Anwendungen miteinander zu verknüpfen. Dies kann auf funktionaler Ebene geschehen, um z.B. von einer Anwendung aus Funktionen einer anderen Anwendung zu nutzen, auf Datenebene, wenn es z.B. darum geht die gleiche Datenbasis zu verwenden, oder auf Geschäftprozessebene, wenn mit mehreren Anwendungen ein Geschäftprozess

abgebildet wird. Will man Anwendungen miteinander verknüpfen, dann müssen sie auch interoperabel sein, denn „Interoperabilität beschreibt die Möglichkeiten von Informations- und Kommunikationssystemen, Daten, Informationen und Wissen miteinander auszutauschen..." [1].

Die Verknüpfung von Anwendungen beinhaltet auch, dass sie stärker voneinander abhängig gemacht werden. Damit aber Unabhängigkeit von Fachanwendungen und der Wunsch nach Integration sich nicht ausschließen, ist es Aufgabe dieses Abschnitts, einen Ausblick auf die kommenden Herausforderungen zu geben, die durch die Forderung nach stärkerer Integration und damit auch stärkerer Interoperabilität für das Themenumfeld Plattformunabhängigkeit zu bewältigen sind.

3.3.1 Integration von Fachanwendungen

Wie in Abschnitt 2.1 beschrieben, bestehen für jede Fachanwendung mehr oder minder starke Abhängigkeiten zu der sie umgebenden IT-Infrastruktur. Ein Teil dieser IT-Infrastruktur ist die Anwendungslandschaft. Sie beschreibt alle Anwendungen und Fachanwendungen der Infrastruktur und deren Beziehungen zueinander.

Die einzelnen Anwendungen der Infrastruktur können in höherem oder geringerem Grade von einer solchen Anwendungsplattform abhängen. Wie bei allen anderen Abhängigkeiten gilt auch hier: Durch Abhängigkeiten werden die Möglichkeiten der Weiterentwicklung einer Umgebung reduziert. Innerhalb einer Anwendungslandschaft kann das weit reichende Folgen haben. Letztlich kann das Hinzufügen oder Ändern einer Anwendung dazu führen, dass andere Anwendungen angepasst oder sogar vollständig ersetzt werden müssen.

Ist beispielsweise eine Anwendung A von einem bestimmten Dienst abhängig, der durch eine Anwendung B zur Verfügung gestellt wird, dann muss sie Zugriff auf diesen haben. Soll nun Anwendung B durch eine andere – beispielsweise kostengünstigere – Alternative ersetzt werden, muss sichergestellt werden, dass die neue Anwendung auch diesen Dienst zur Verfügung stellt. Solche Abhängigkeiten stellen für die strategische Planung von IT-Infrastrukturen im Allgemeinen bzw. von Anwendungsplattformen im Speziellen eine große Herausforderung dar.

Zum einen wird die Interoperabilität von Anwendungen gefordert, um so die Konsolidierung der IT-Landschaft zu vereinfachen. Zum anderen wäre es wichtig, die Abhängigkeiten auf ein Minimum zu reduzieren, um eine maximale Auswahl an Alternativen für die Weiterentwicklung zu erhalten. Spätestens hier wird klar,

[1] European Interoperability Framework for Pan-European eGovernment Service, Version 1.0, November 2004 (siehe auch Glossar „Interoperabilität").

dass Integration und Plattformunabhängigkeit zwei Ziele sind, die sich nicht ohne weiteres vereinbaren lassen.

Ein Extrembeispiel für eine Anwendungsplattform mit minimalen Abhängigkeiten sind die so genannten Silo-Architekturen. In einer solchen Architektur stehen die Anwendungen unverbunden und autark wie Silos nebeneinander.

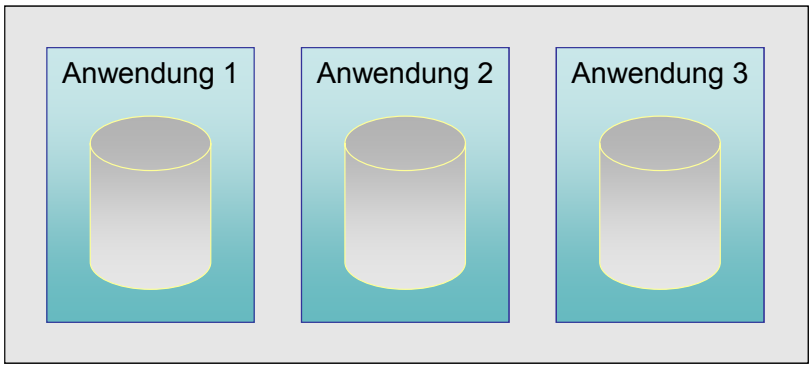


Abbildung 9: Silo-Architektur

In einer Silo-Architektur arbeiten die einzelnen Anwendungen unabhängig voneinander. Jede Anwendung realisiert die von ihr abzubildenden Prozesse vollständig autark und ist daher nicht auf andere Anwendungen angewiesen. In einer Silo-Architektur sind die einzelnen Anwendungen daher unabhängig von der sie umgebenden Anwendungsplattform. Soll eine der Anwendungen verändert oder ersetzt werden, stehen die anderen Anwendungen dem nicht entgegen, da sie von der Veränderung nicht betroffen sind. Aus dem Blickwinkel der Plattformunabhängigkeit betrachtet, bilden Silo-Architekturen eine schon fast optimale Lösung.

Allerdings gilt es zu beachten, dass Anwendungsplattformen, die auf Silos setzen, davon ausgehen, dass prinzipiell jede Aufgabe auch ohne Integration lösbar ist. Dabei werden zum Teil erhebliche Nachteile akzeptiert. Z. B. ist es nicht möglich, Bestandteile einer Silo-Anwendung in einer anderen Anwendung wieder zu verwenden. Stattdessen müssen die betreffenden Anwendungsteile zweimal implementiert werden. Dafür bleibt die grundsätzliche Lauffähigkeit der Silo-Anwendungen in jedem Fall erhalten.

Viele Prozesse sind jedoch so komplex, dass sie nicht durch eine einzelne Anwendung vollständig abgebildet werden können. Vielmehr sind verschiedene Anwendung an der informationstechnischen Umsetzung des Prozesses beteiligt. Als Beispiel diene hier ein gebührenpflichtiges Antragsverfahren, das von einer Behörde als Online-Dienstleistung angeboten wird. An der Umsetzung eines solchen Verfahrens könnten unter anderem folgende Komponenten beteiligt sein:

- Eine Web-Anwendung, über die ein Bürger einen Antrag stellen kann.
- Ein Dokumenten-Management-System (DMS), in dem der gestellte Antrag gespeichert wird.
- Eine Anwendung, die dafür zuständig ist, die Zahlung der Gebühr durch den Antragsteller zu veranlassen.

Stellt nun ein Bürger einen Antrag über die Web-Anwendung, muss diese dafür sorgen, dass der Antrag im DMS abgelegt wird. Erreicht das Verfahren einen Zustand, in dem die Gebühr fällig ist, muss das DMS oder eine weitere Komponente diejenige Anwendung, die für die Veranlassung der Zahlung zuständig ist, über diesen Zustandseintritt informieren.

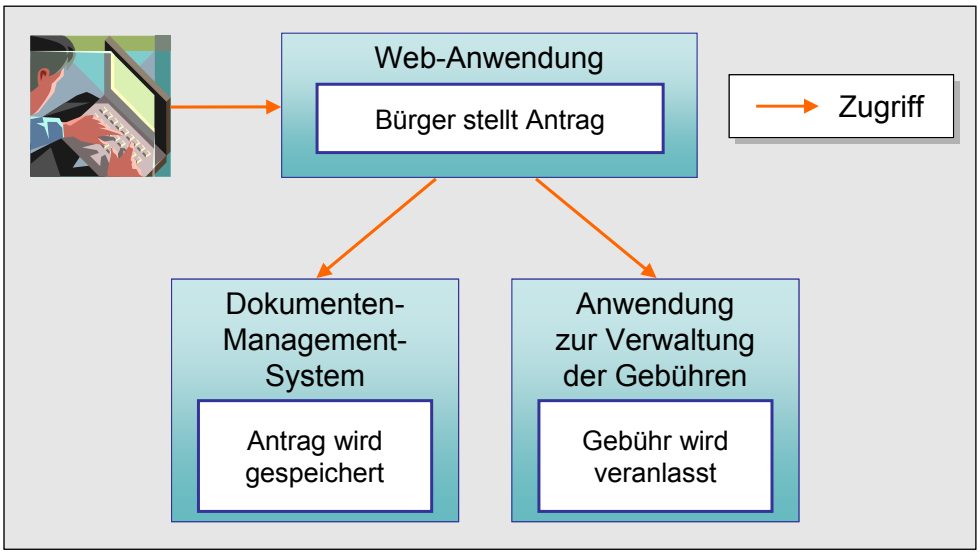


Abbildung 10: Integrierte Fachanwendung

Wie das Beispiel zeigt, müssen die an der gemeinsamen Realisierung eines Prozesses beteiligten Anwendungen integriert werden, um Medienbrüche oder unnötige Wartezeiten zu vermeiden. Es gibt also offensichtlich Fälle, in denen es nicht ohne Integration geht. Im weiteren Verlauf dieses Kapitels werden daher Hinweise gegeben, wie eine Integration durchgeführt werden kann, ohne dass der Gedanke der Plattformunabhängigkeit vollständig aufgegeben wird.

Diese Hinweise erfolgen bewusst in recht abstrakter Form. Die Integration von Anwendungen ist ein hoch komplexes Thema, dessen tiefere Betrachtung den Rahmen dieses Leitfadens deutlich sprengen würde. Für interessierte Leser sei daher u. a. auf das e-Government-Handbuch des Bundesamtes für Sicherheit in

der Informationstechnik (BSI)[1] verwiesen, welches sich intensiv mit dem Thema Integration im öffentlichen Umfeld auseinander setzt.

Aus Sicht der Plattformunabhängigkeit tritt die erste Herausforderung bei der Anwendungsintegration durch die Art und Weise auf, wie Integrationen zustande kommen. In der Praxis geschieht dies häufig eher bedarfsgetrieben und unterliegt daher keinem geordneten Prozess. Integrationen werden ad-hoc vorgenommen und verbinden meist zwei Anwendungen direkt miteinander. Es entstehen so genannte Punkt-zu-Punkt-Integrationen. Die Anwendungen kommunizieren miteinander ohne Zuhilfenahme von Integrationskomponenten. Punkt-zu-Punkt-Integrationen sind nicht grundsätzlich problematisch, werden sie jedoch in großem Umfang eingesetzt, führen sie zu einer so genanten „Stovepipe-Architektur[2]“, bei der jedes Element mit jedem kommuniziert.

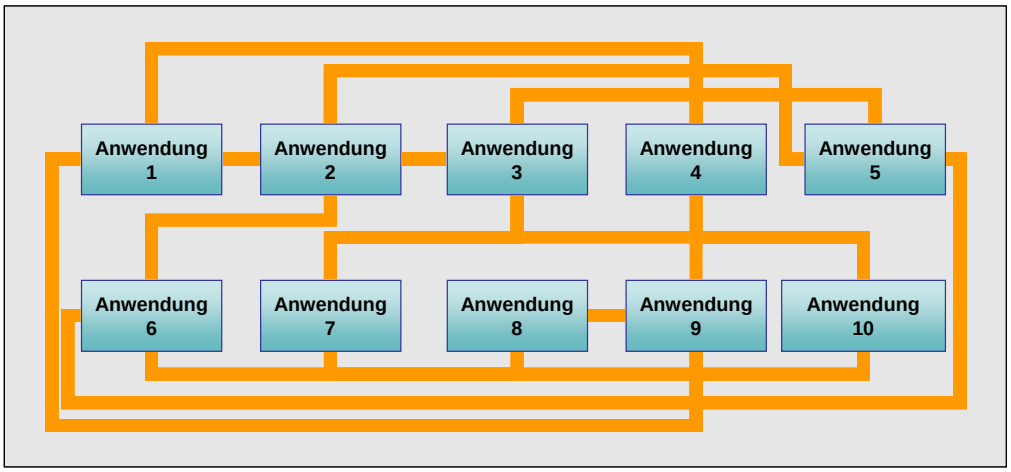


Abbildung 11: Stovepipe-Architektur

Stovepipe-Architekturen führen zu einem hohen Grad an Plattformabhängigkeit der beteiligten Anwendungen. Sie sind insofern das Gegenstück zu den Silo-Architekturen. Soll eine der Anwendungen geändert oder ersetzt werden, müssen alle Anwendungen berücksichtigt werden, die mit dieser Anwendung verbunden sind. Da das häufig sehr viele Anwendungen sind, ist der Spielraum sehr gering bzw. der Anpassungsaufwand sehr hoch.

Folglich muss unter Aspekten der Plattformunabhängigkeit schon bei den ersten Anwendungen, die integriert werden, darauf geachtet werden, dass direkte Punkt-zu-Punkt-Integrationen nur dann verwendet werden sollten, wenn gute Gründe dafür sprechen (siehe auch 4.1.3.3).

1 siehe http://www.bsi.de/fachthem/egov/3.htm

2 Sinngemäß übersetzt: Ofenrohr-Architektur

Die gemeinsame Betrachtung der Themen Integration und Plattformunabhängigkeit lässt sich vereinfachend auf einen Konfliktpunkt reduzieren:

> *Die Forderung nach Interoperabilität steht gegen die Forderung nach Unabhängigkeit. Dieser Konflikt lässt sich nicht auflösen, aber zumindest reduzieren.*

Technisch gesehen ist die Forderung nach Interoperabilität eine Forderung nach Schnittstellen zwischen Anwendungen. Wie in Abschnitt 2.1 beschrieben, entstehen Abhängigkeiten gerade an diesen Schnittstellen. Umso wichtiger ist es, beim Design der Schnittstellen darauf zu achten, dass diese möglichst zukunftssicher und flexibel gestaltet werden. Damit verbietet sich der Einsatz von proprietären Protokollen von selbst, die nur von exakt den an der Schnittstelle beteiligten Anwendungen verstanden werden. Ein Austausch einer der beiden Anwendungen wäre in diesem Fall kaum noch möglich, insbesondere wenn das Schnittstellenprotokoll nicht sauber dokumentiert wurde.

Anders verhält es sich, wenn ein zukunftssicheres Protokoll basierend auf offenen Standards verwendet wurde. Solche Standardprotokolle haben in der Regel einen hohen Verbreitungsgrad, werden also von vielen Anwendungen verstanden. Nutzt eine Anwendung eine Schnittstelle unter Verwendung eines Standardprotokolls und soll die anbietende Anwendung ersetzt werden, so ist das in den meisten Fällen möglich, da die Schnittstelle auch mit einer Vielzahl anderer Produkte realisiert werden kann. Die nutzende Anwendung muss in diesem Fall kein neues Protokoll implementieren, da die Schnittstelle bezüglich des Protokolls unverändert bleibt.

Wird dagegen nur ein spezielles proprietäres Protokoll eingesetzt, erschwert dies eine Ablösung der anbietenden Anwendung, da dann nur Produkte in Betracht kommen, die ebenfalls dieses proprietäre Protokoll verstehen. Handelt es sich um ein Protokoll, dass nur von Produkten eines bestimmten Herstellers verwendet wird, entsteht folglich eine Herstellerabhängigkeit.

Ähnliches gilt für das Format, in dem Daten zwischen den Anwendung ausgetauscht werden. Auch hier können sowohl auf Standards basierende Datenaustauschformate – beispielsweise auf XML –, als auch proprietäre Formate – meist spezielle Binärformate – verwendet werden. Genau wie die Standardprotokolle bieten Datenaustauschformate, die auf Standards basieren, ein höheres Maß an Unabhängigkeit, vor allem in Bezug auf die Herstellerunabhängigkeit aber i.d.R. auch in Bezug auf die Plattformunabhängigkeit. Darüber hinaus kann durch die Verwendung von standardisierten Datenaustauschformaten Interoperabilität sichergestellt werden. Demgegenüber schafft die Verwendung proprietärer Datenformate meist höhere Plattform- und vor allem auch Herstellerabhängigkeit.

3.3.2 Fazit zu Integration und Plattformunabhängigkeit

In Abschnitt 0 wurde gezeigt, dass Fachanwendungen als Bestandteil der gesamten IT-Infrastruktur einer Behörde nur selten isoliert betrachtet werden können. Gerade Themen wie die Konsolidierung und Integration von IT-Infrastrukturen stellen hohe Anforderungen an die Interoperabilität von Fachanwendungen. Solche Anforderungen können sich durchaus mit der Forderung nach plattformunabhängigen Fachanwendungen widersprechen.

Diesen Widerspruch kann man nicht auflösen, aber zumindest deutlich reduzieren. So ist im Rahmen des strategischen Abhängigkeitsmanagement frühzeitig darauf zu achten, dass Punkt-zu-Punkt-Integrationen nur dann verwendet werden sollten, wenn gute Gründe dafür sprechen. Ein weiterer wichtiger Aspekt ist, dass bei der Umsetzung von Fachanwendungen auf den Einsatz von offenen Standards und offenen Schnittstellen geachtet wird. Dadurch lassen sich die Abhängigkeiten, die durch die Interoperabilität bei der Integration entstehen, zumindest mindern.

Insgesamt verhindern diese Maßnahmen die Plattformabhängigkeit zwar nicht, sorgen aber dafür, dass sie bei den entstehenden integrierten Anwendungsplattformen nur schwach ausgeprägt sein wird.

4 Entwicklung und Beschaffung (plattform)unabhängiger Fachanwendungen

In den Eingangskapiteln wurde aufgezeigt, dass Abhängigkeiten in Fachanwendungen immer dann ein Problem sind, wenn sie notwendige Veränderungen beoder gar verhindern. Ferner wurde dargelegt, dass die Frage plattformunabhängig oder plattformabhängig häufig auch eine strategische Dimension beinhaltet, nämlich immer dann, wenn Fachanwendungen längerfristig eingesetzt werden oder wenn eine Vielzahl von Fachanwendungen eingesetzt wird. Beide Punkte stehen in enger Verbindung miteinander. Veränderungen an einem gegebenen Zustand bezogen auf eine Fachanwendung oder auf eine ganze Anwendungslandschaft lassen sich technisch sicherlich in den meisten Fällen durchführen. Es muss jedoch auch die Frage nach der Wirtschaftlichkeit der geplanten Änderungen beantwortet werden. Mit anderen Worten: Welcher Aufwand ist mit den Änderungen verbunden und steht dieser Aufwand noch in Relation zum gewünschten Nutzen?

Wie schon in den vorangegangen Kapiteln aufgezeigt lassen sich drei Fälle unterscheiden, wie eine Verwaltung zu einer benötigten Fachanwendung kommen kann. Dies sind

- die Entwicklung innerhalb der eigenen Organisation mit eigenem Personal (Eigenentwicklung),
- die Beauftragung der Entwicklung (Auftragsentwicklung) und
- der Kauf[1] einer Fachanwendung.

Wenn im Folgenden von einer Beschaffung die Rede ist, dann steht dies synonym für diese drei Möglichkeiten.

Um die Kosten für Veränderungen minimal zu halten und damit die Wirtschaftlichkeit zu verbessern, ist es sinnvoll, diese frühzeitig, das heißt, bereits im Vorfeld der Beschaffung der Fachanwendung zu berücksichtigen. Für alle drei Fälle, Eigenentwicklung, Auftragsentwicklung und Kauf, muss die Wirtschaftlichkeit der Beschaffung gewährleistet sein, egal ob nun plattformunabhängig oder plattformabhängig. Vielmehr ist es sogar so, dass im Rahmen einer Wirtschaftlichkeitsbetrachtung im Vorfeld der eigentlichen Beschaffung zweierlei zu prüfen ist,

[1] Die Beschaffung kann aber auch noch über Miete, Leasing oder einen Applikation Service Provider (ASP) erfolgen. Im Folgenden umfasst die Erwähnung der Möglichkeit der Beschaffung per Kauf auch immer die drei anderen Möglichkeiten.

- erstens ob die Beschaffung als Eigenentwicklung[1], als Auftragsentwicklung oder als Kauf[2] wirtschaftlicher ist und
- zweitens, ob eine plattformunabhängige Lösung wirtschaftlicher ist als eine plattformabhängige Lösung.

Wobei diese zwei Ebenen der Prüfung natürlich nicht losgelöst voneinander zu betrachten sind, denn sowohl bei der Eigenentwicklung, der Auftragsentwicklung als auch beim Kauf stellt sich immer die Frage: plattformabhängig oder plattformunabhängig? Näheres hierzu wird im Kapitel 4.2.2 erläutert.

Das Maß der Unabhängigkeit oder Abhängigkeit einer Fachanwendung von ihrer Ablaufumgebung spielt immer dann eine Rolle, wenn die Fachanwendung oder auch die Ablaufumgebung Änderungen unterworfen ist. Solche Änderungen können wirtschaftlich, technisch oder auch fachlich bedingt sein.

Wirtschaftlich bedingte Veränderungen können zum Beispiel notwendig werden, wenn man sich von einem Hersteller (z.B. von einem Datenbankmanagementsystem oder einem Betriebssystem) lösen will, weil dessen Lizenzpolitik die laufenden Kosten für das jeweilige Produkt erheblich in die Höhe treibt. Solchen wirtschaftlichen Ursachen kann man u.a. durch langfristige Verträge entgegenwirken. Andererseits schließt man mit solchen Verträgen aber ggf. auch die Mitnahme von positiven Marktentwicklungen aus, wie z.B. Kostensenkungen.

Technisch bedingte Änderungsursachen liegen häufig in der Weiterentwicklung sowohl der Fachanwendung als auch ihrer Ablaufumgebung. So werden von eigentlichen allen Herstellern von Betriebssystemen, Datenbankmanagementsystemen oder sonstigen relevanten Standardanwendungen sowie vielen Infrastrukturdiensten in regelmäßigen Abständen neue Versionen zur Verfügung gestellt. Solange der Support der alten Versionen durch den Hersteller noch gewährleistet ist, ist eine Anpassung in der Ablaufumgebung der Fachanwendungen meist nicht notwendig, es sei denn, man möchte neue Funktionen der neuen Version nutzen.

Fachlich bedingte Änderungen an Fachanwendungen werden in der öffentlichen Verwaltung häufig durch Gesetzesvorgaben bzw. -änderungen ausgelöst. In der Regel sind dies meist nicht nur rein funktionale Änderungen sondern häufig auch Änderungen im Prozessablauf, die in die Fachanwendung integriert werden müssen. Dies insbesondere deshalb, weil heute und zukünftig immer mehr Verwaltungsprozesse einer Automatisierung zugeführt werden, was gleichzeitig auch eine sehr viel stärkere Verbindung von einzelnen Fachanwendungen mit sich bringt. Gute Beispiele hierfür sind auf der Ebene der Querschnittsaufgaben das Zusammenwirken von Personalverwaltung, Zeiterfassung, Zugangskontrolle, Ur-

[1] Dies gilt natürlich nur dann, wenn für die Organisation überhaupt die Möglichkeit einer Eigenentwicklung besteht.

[2] Vorausgesetzt, es gibt am Markt überhaupt fertige Lösungen.

laubsworkflow u.a. oder auf der Ebene der Fachaufgaben das geplante Zusammenwirken der vielen verschiedenen Fachverfahren aller Meldebehörden ab 2007. Je länger eine Fachanwendung eingesetzt werden soll und je größer die Zahl der Anwendungen, die miteinander kommunizieren (egal ob Fachanwendung mit Fachanwendung oder Fachanwendung mit anderen Anwendungen), umso größer ist die Wahrscheinlichkeit, dass eine Veränderung notwendig und damit auch das Maß der Abhängigkeit hinsichtlich Aufwand und Wirtschaftlichkeit eine ausschlaggebende Rolle spielen wird. Ein weiterer Aspekt in diesem Zusammenhang ist die Menge der Fachanwendungen, die, wie die Erfahrungen gezeigt haben, bei fehlender Unabhängigkeit zu erheblichen Kosten bei notwendigen Änderungen führen können. So hat z.B. die Abhängigkeit einer Vielzahl von Fachanwendungen von dem Betriebssystem Windows bei Behörden, die viele solcher Fachanwendungen einsetzen, den Wechsel nach Linux aus Kostengründen vorerst verhindert. Die somit entstandenen Herstellerabhängigkeit macht deutlich, dass der Umgang mit Abhängigkeiten im Zusammenhang von Fachanwendungen und ihrer Ablaufumgebung eine immer größere strategische Dimension besitzt, je mehr Fachanwendungen länger eingesetzt werden und umso größer die Komplexität des Zusammenspiels aller Anwendungen ist.

Daher wird empfohlen, die gesamte Thematik der Abhängigkeiten von Fachanwendungen als strategische Herausforderung zu betrachten, der sich eine Behörde stellen muss. Sie sollte daher innerhalb ihrer IT-Strategie eine IT-Gesamt- oder Enterprisearchitektur[1] mit Architekturvorschriften zur Entwicklung der Architektur der jeweiligen Fachanwendung festschreiben. Im Kapitel 4.1 werden hierzu nähere Ausführungen gemacht.

Im Folgenden wird im ersten Schritt deutlich gemacht, wie dem Problem von Abhängigkeit bei Fachanwendungen aus strategischer Sicht begegnet werden kann. Die strategischen Aspekte nehmen dabei eine dem eigentlichen Beschaffungsprozess übergeordnete Position ein. Anschließend wird das Thema aus dem Blickwinkel des eigentlichen Beschaffungsprozesses unter Berücksichtigung der strategischen Aspekte beleuchtet.

[1] Der Begriff "Architektur" wird besonders im Zusammenhang mit Softwaresystemen auf vielerlei Weise verwendet und mit einer solchen Vielfalt von Bedeutungen belegt, dass nicht alle von ihnen immer miteinander vereinbar sind. Es empfiehlt sich daher stets, die Bedeutung von "Architektur" im gegebenen Kontext zu definieren. Für das vorliegende Dokument gilt dies insofern, als es über unterschiedliche Architekturen spricht – die IT-Gesamtarchitektur und die tatsächliche Architektur der zu beschaffenden Anwendung (Anwendungsarchitektur).

4.1 Strategische Aspekte

4.1.1 Strategisches Abhängigkeitsmanagement

Im Mittelpunkt des Kapitels 3 standen Begriffe und Muster von Anwendungsarchitekturen sowie Technologien, mit denen Anwendungsarchitekturen realisiert werden können. Die Empfehlungen für die Verwendung dieser Technologien leiteten sich aus zwei Motiven ab:

- aus den qualitativen Merkmalen der architektonischen Begriffe und Muster, namentlich der Plattformunabhängigkeit, und
- aus den fachlichen Anforderungen an die Anwendung.

Damit bietet das vorausgegangene Kapitel das grundlegende Rüstzeug für eine selten anzutreffende Situation: die Entwicklung einer einzigen, isolierten Fachanwendung mit der großen Gestaltungsfreiheit der so genannten "Grünen Wiese".

In der Praxis fließt ein weiteres Motiv in die Gestaltung von Fachanwendungen ein: nämlich die Beschränkungen des konkreten Einzelfalles. Sie sind zentrale Gegenstände dieses Kapitels, die dabei helfen, die Situation von der "grünen Wiese" in einen konkreten Software-Anwendungsfall zu überführen.

Beschränkungen lassen sich ihrer Herkunft nach in drei Gruppen bündeln:

- Beschränkungen, die auf Vorgehensweisen oder früher getroffene Entscheidungen der Organisation zurückgehen,
- Beschränkungen, die durch die existierende Anwendungslandschaft entstehen und
- Beschränkungen durch den Markt für vorkonfektionierte Lösungen (Commercial Off-The-Shelf'-Software (COTS)).

Da die Marktsituation praktisch nicht unmittelbar beeinflussbar ist, wird sie in diesem Kapitel nur am Rande betrachtet. Die Anwendungslandschaft und die Organisation sind jedoch gestaltbar. Auf sie richtet sich der Fokus dieses Kapitels.

Die Organisation spielt die zentrale Rolle. Plattformabhängigkeit wird dann als Problem sichtbar, wenn ein Teil der Umgebung sich ändern soll oder muss, und eine IT-Landschaft nicht oder nur schwer in der Lage ist, diese Änderung mit zu vollziehen. Eine solche Situation lässt sich im Regelfall auf eine Reihe von Entscheidungen in der Vergangenheit zurückführen, man spricht nicht umsonst von „historisch gewachsenen Strukturen“. Um zu vermeiden, dass Plattformabhängigkeit zum Problem wird, müssen Entscheidungen also mit Weitsicht getroffen werden, was die Bedeutung einer langfristigen Strategie nochmals unterstreicht.

Anwendungslandschaften verdeutlichen und verschärfen das Problem, das die Organisation bewältigen muss – die Umsetzung von Geschäftsprozessen. Die Unterstützung der Geschäftsprozesse einer Organisation durch die IT reicht idea-

ler Weise bruchlos von einem Ende der Organisation zum anderen. Der konsequente Einsatz von Fachanwendungen führt daher in der Regel automatisch dazu, dass sie untereinander immer enger integriert werden und die einzelne Anwendung als Betrachtungsgegenstand durch eine integrierte Anwendungslandschaft abgelöst wird. Wie das vorausgegangene Kapitel 3 gezeigt hat, stehen Anwendungsintegration und (Plattform)Unabhängigkeit im Widerspruch zueinander. Unabhängigkeit von Fachanwendungen von ihrem Basis-System ist daher immer auch eine Frage einer weitsichtigen (strategischen) Abwägung zwischen gleichermaßen legitimen, jedoch widerstreitenden Interessen auf dem Weg zum übergeordneten Ziel der Wirtschaftlichkeit.

Die Aufgabe der Organisation im Angesicht ihrer existierenden oder kommenden Anwendungslandschaften ist somit dauernd und kann zusammenfassend als "strategisches Abhängigkeitsmanagement" charakterisiert werden. Für die Bewältigung dieser Aufgabe enthält das Kapitel 4.1 eine Reihe von Empfehlungen, und an ihr orientiert sich die Betrachtung der Gegebenheiten in der Organisation sowie in der Anwendungslandschaft.

4.1.2 Prozess zur Gestaltung der IT-Gesamtarchitektur

Die Plattformunabhängigkeit sollte in der IT-Strategie jeder Behörde verankert werden. Die Priorität, die das Ziel der Unabhängigkeit von Fachanwendungen in der jeweiligen IT-Strategie einnehmen soll, ergibt sich im Rahmen des strategischen Gestaltungsprozesses „automatisch“ aus Umfang und Komplexität der Anwendungslandschaft der jeweiligen Behörde. Gleichsam sollte sich dabei auch Umfang und Komplexität der jeweils festzulegenden IT-Gesamtarchitektur sowie der entsprechenden Architekturvorschriften (siehe auch Kapitel 4.1.2.1) ergeben.

Die Begriffe und Muster von Anwendungsarchitekturen wurden in Kapitel 3 überwiegend mit einem relativ engen Fokus darauf vorgestellt, was sie zur Strukturierung und zum Funktionieren einer einzelnen Anwendung beitragen. Eine Aufgabe des strategischen Abhängigkeitsmanagements ist es, die Anwendung dieser Muster und Technologien über den Rahmen der einzelnen Anwendung hinaus zu bedenken und zu definieren. Das Ergebnis dieser Definition ist die IT-Gesamtarchitektur sowie die dazugehörigen Architekturvorschriften.

Eine zentrale Funktion der IT-Gesamtarchitektur ist die strategische Steuerung der Entwicklung der Anwendungslandschaft. "Strategisch" identifiziert in diesem Zusammenhang alle diejenigen Aspekte der IT-Gesamtarchitektur, die nicht aus den Anforderungen der individuellen Anwendungen abgeleitet werden können oder sollen. Einer dieser Aspekte ist Plattformunabhängigkeit. Solche strategischen Ziele wiegen prinzipiell schwerer als anwendungsspezifische Anforderungen. Wenn die IT-Gesamtarchitektur beispielsweise vorsieht, dass Office-Dokumente, die von Anwendungen generiert oder zwischen ihnen ausgetauscht werden, in einem plattformunabhängigen Standardformat vorliegen müssen,

dann ist diese Forderung ein wesentliches Auswahlkriterium für alle Einzelanwendungen.

Die IT-Gesamtarchitektur wird in diesem Sinne in erster Linie als Referenzmodell für individuelle Gestaltung der Anwendungsarchitektur verwendet.

Dabei gibt es zwei Gestaltungsfelder, die durch eine IT-Gesamtarchitektur beschrieben und gelenkt werden sollten:

- die Evolution der einzelnen Anwendungen und
- die Integration der Anwendungen untereinander.

Jedoch noch vor der Frage, womit eine IT-Gesamtarchitektur befüllt wird (diese Frage wird in Abschnitt 4.1.3 behandelt) muss eine andere Frage beantwortet werden: Wie kommt eine IT-Gesamtarchitektur zustande, wie wird sie gepflegt, und vor allem, wie soll sie verwendet werden?

Der Prozess zur Gestaltung der IT-Gesamtarchitektur über die Anwendungslandschaft einer Behörde orientiert sich aufgrund der strategischen Steuerungsfunktion, die sie einnehmen soll, am Prozess zur Konzeption, Umsetzung und Fortschreibung der jeweiligen IT-Strategie. Dieser sieht i.d.R. wie folgt aus (siehe auch Abbildung 12):

- Entwicklung eines Leitbildes auf Basis des Auftrages der jeweiligen Organisation bzw. der zu erbringenden Dienstleistungen und auf Basis der Ergebnisse einer Umfeldanalyse.
- Im Folgeschritt - Entwicklung der eigentlichen IT-Strategie mit ihren Anforderungen an verschiedene Dimensionen des IT-Bereiches, wie zum Beispiel den Finanzen, den Kunden, der Innovation usw., sowie ihren Produkten. Ein Teil dieser Produkte sind die IT-Gesamtarchitektur sowie die dazugehörigen Architekturvorschriften, die sich aus Architekturvorschriften der IT-Strategie des Bundes[1] ableiten.
- Im Anschluss an die Konzeption der IT-Strategie erfolgt dann die Umsetzungsplanung, die Umsetzung und die regelmäßige Fortschreibung der IT-Strategie.

[1] Derzeit SAGA in der Version 3.0

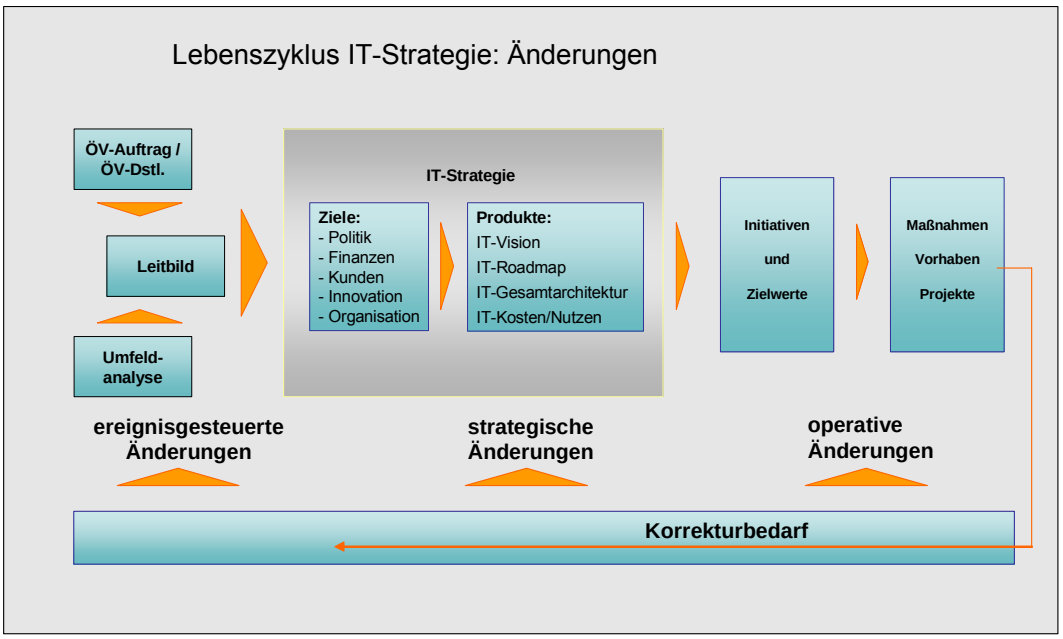


Abbildung 12: IT-Strategieprozess

Der entsprechende, in der nachfolgenden Abbildung 13 dargestellte Prozess zur Gestaltung der IT-Gesamtarchitektur hat zum Ziel, die aktuell bestehende Anwendungslandschaft ausgerichtet an den Zielen der IT-Strategie insgesamt und der definierten IT-Gesamtarchitektur im besonderen in eine den Anforderungen der zu bewältigenden Verwaltungsaufgaben angemessene Ziel-Anwendungslandschaft zu überführen.

Dieser strategische Gestaltungsprozess untergliedert sich in vier wesentliche Teilschritte:

- Schritt 1 - Entwicklung der IT-Gesamtarchitektur
- Schritt 2 - Planung der Umsetzung der IT-Gesamtarchitektur
- Schritt 3 - Umsetzung der IT-Gesamtarchitektur
- Schritt 4 - Pflege und Fortschreibung

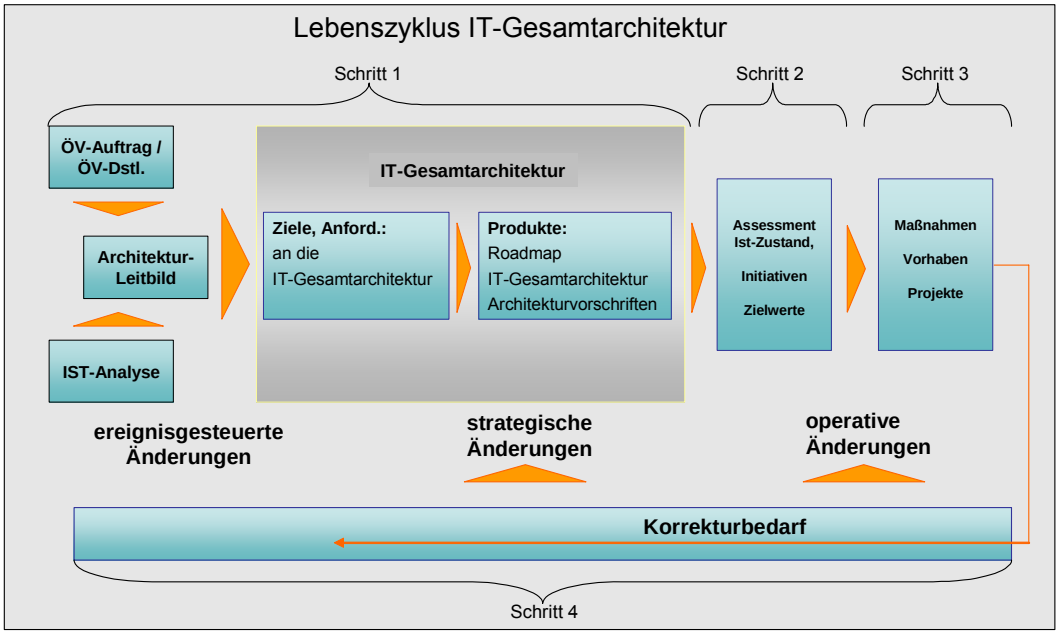


Abbildung 13: Strategischer Gestaltungsprozess „IT-Gesamtarchitektur

4.1.2.1 Schritt 1 - Entwicklung der IT-Gesamtarchitektur

Das wichtigste Motiv hinter dem Einsatz von Fachanwendungen ist die Umsetzung und die Automatisierung von Geschäftsprozessen. Dies führt in der Mehrzahl aller Fälle fast automatisch zur Integration von Fachanwendungen. Dementsprechend sind Fachanwendungen im Zusammenhang von "Anwendungslandschaften" zu verstehen. Deren Entwicklung ist mit denen der individuellen Fachanwendungen zu verzahnen, und sie sind zugleich Instrument zur Durchsetzung nichtfunktionaler Merkmale wie z.B. Plattformunabhängigkeit. Mit Blick auf die zuvor genannten Probleme bietet die Vereinheitlichung einer solchen Anwendungslandschaft zudem die Möglichkeit, die beschränkten Kräfte effektiv zu bündeln.

In diesem Sinne berücksichtigt die IT-Gesamtarchitektur die zu automatisierende Verwaltungsprozesse einer Behörde, verbindet zwei Softwarearchitekturen miteinander und formuliert anwendungsübergreifende Aussagen.

Beispiel 1 Beispiel: Die IT-Gesamtarchitektur sieht ein synchrones Modell der Kommunikation zwischen Anwendungen vor. Eine ideale Architektur einer Fachanwendung losgelöst von einer angestrebten Anwendungslandschaft geht jedoch von einem asynchronen, Polling-basierten Kommunikationsmodell aus. Um mit der IT-Gesamtarchitektur zu harmonieren, muss die Anwendungsarchitektur auf ein synchrones Modell umgestellt werden. Gelingt dies nicht, weil die Fachanwendung die damit verbundenen Latenzzeiten nicht bewältigen kann, so muss sie um eine puffernde Kommunikationskomponente erweitert werden.

Ein weiterer wesentlicher Aspekt der IT-Gesamtarchitektur sind die dazugehörigen Architekturvorschriften. Diese sind aus den Architekturvorschriften zur IT-Strategie des Bundes „SAGA“[1] abzuleiten und ggf. gemäß den Bewertungskriterien von SAGA um fehlende Vorschriften zu ergänzen. Die Architekturvorschriften beinhalten zu verwendende Standards, Technologien und Architekturmuster. Im Gegensatz zu SAGA sollen die Architekturvorschriften einer Behörde mit Bezug auf die IT-Gesamtarchitektur schon genau festlegen wann, wie und wo welche Vorschrift anzuwenden ist. Dies erleichtert dann im späteren Beschaffungsprozess auch die Formulierung konkreter Anforderungen zur Einhaltung der SAGA-Konformität.

In diesem Kontext ist es nicht Ziel dieses Dokumentes, dem Leser aufzuzeigen, wie er nun von den Aufgaben seiner Behörde zu einer vollständigen IT-Gesamtarchitektur kommt. Hierfür gibt es schon ausreichend geeignete Literatur auf dem Markt. Ziel und Aufgabe dieses Dokumentes ist es, dem Leser Hinweise und Empfehlungen zur Gestaltung der IT-Gesamtarchitektur hinsichtlich der Wahrung eines angemessenen Maßes an Unabhängigkeit zu geben. Dies erfolgt ausführlich in Kapitel 4.1.3.

4.1.2.2 Schritt 2 - Planung der Umsetzung der IT-Gesamtarchitektur

Eine der wichtigsten Maßnahmen in diesem Schritt ist die Durchführung eines Assessments der bestehenden Anwendungslandschaft gemessen an den Vorgaben der IT-Gesamtarchitektur und den vorliegenden Architekturvorschriften, um daraus entsprechende Initiativen und Zielwerte für die Weiterentwicklung der bestehenden Anwendungslandschaft abzuleiten. Hinsichtlich der Bewertung der Abhängigkeitsproblematik innerhalb dieses Assessments liefern die Kapitel 2, 3 und 4.1.3 die notwendigen Hinweise.

4.1.2.3 Schritt 3 - Umsetzung der IT-Gesamtarchitektur

Im dritten Schritt werden nun aus den Initiativen und Zielwerten entsprechende Maßnahmen, Vorhaben und Projekte zur konkreten Umsetzung der angestrebten Anwendungslandschaft definiert. Das können Maßnahmen zur Anpassung der bestehenden Anwendungslandschaft oder Neubeschaffungen von Fachanwendungen sein, um das Portfolio der automatisierten Verwaltungsprozesse zu vervollständigen. Im Rahmen dieser Beschaffungsmaßnahmen sind dann auch die entsprechenden Anforderungen zur Sicherstellung der notwendigen Unabhängigkeit für die konkrete Fachanwendung zu formulieren.

[1] siehe http://www.kbst.bund.de/-,182/SAGA.htm

Der Beschaffungsprozess von Fachanwendungen gliedert sich dann in drei Stufen:

- Zunächst werden auf Basis der fachlichen Anforderungen die technischen Anforderungen einer Anwendung ermittelt und die daraus resultierende Anwendungsarchitektur definiert. Diese nimmt die neu entwickelten oder möglicherweise erweiterten technischen Maßgaben der IT-Gesamtarchitektur auf, ebenso wie deren Gewichtung.
- Als nächstes ist die Entscheidung zu treffen, ob Vorhandenes gekauft oder Neues entwickelt werden muss (make or buy). Diese Entscheidung muss sowohl auf technischen als auch auf wirtschaftlichen Aspekten beruhen. Die durchzuführende Maßnahme ist hier eine Wirtschaftlichkeitsbetrachtung nach WiBe 4.0 (siehe Kapitel 4.2.2).

  Insbesondere beim Kauf kann die Marktlage unter Umständen dazu führen, Kompromisse eingehen zu müssen. Geraten diese Kompromisse allerdings in Konflikt mit den essenziellen Maßgaben der strategischen Architektur, dann ist zu prüfen, ob nicht anstelle des Kaufes eine Entwicklung wirtschaftlicher ist.
- Zuletzt steht dann die eigentliche Beschaffung der Anwendung an. Sollte der Weg der Beschaffung über eine Ausschreibung erfolgen (Entwicklungsbeauftragung oder Kauf), dann gilt es sowohl aus technischer Sicht als auch aus rechtlicher Sicht weitere Hinweise hinsichtlich der Formulierung von Anforderungen zu berücksichtigen (siehe Kapitel 4.2.1 und 4.2.3).

Im Detail wird hierauf, soweit es das Leitthema dieses Leitfadens betrifft, in Kapitel 4.2 eingegangen.

4.1.2.4 Schritt 4 - Pflege und Fortschreibung

Im letzten Schritt geht es nun um die Pflege und Fortschreibung der IT-Gesamtarchitektur., das heißt, die Anpassung der IT-Gesamtarchitektur an veränderte Gegebenheiten. Anpassungsbedarf kann sich dabei ergeben aufgrund:

- Ereignisgesteuerte Änderungen
  können sich z.B. aufgrund von Aufgabenverlagerungen durch politische Entscheidungen ergeben. Was aber für die IT-Gesamtarchitektur sicherlich noch viel wichtiger und ausschlaggebender ist, sind maßgebliche Technologieänderungen oder –neuerungen. Sofern sie für die Entwicklung der Anwendungslandschaft von Bedeutung sind, sollten diese in IT-Gesamtarchitektur einfließen.

- Strategische Änderungen
  sind von grundlegender Bedeutung und können wirtschaftlich oder politisch motiviert sein. Sie müssen in die IT-Gesamtarchitektur einfließen.
- Operative Änderungen
  können zum Beispiel herbeigeführt werden, wenn sich in einem konkreten Beschaffungsprozess herausstellt, dass konkrete architektonische Vorgaben, die aus der IT-Gesamtarchitektur oder den Architekturvorschriften abgeleitet wurden, grundsätzlich nicht umsetzbar sind.

4.1.3 Wahl der IT-Gesamtarchitektur

Die zukünftige Anforderung an die IT-Landschaft einer Behörde besteht darin, die von der Behörde angebotenen Dienstleistungen vollständig und medienbruchfrei zu unterstützen. Wie bereits in Kapitel 3 dargelegt wurde, können einzelne Fachanwendungen dies in der Regel nicht mehr leisten. Vielmehr müssen Fachanwendungen mit anderen Komponenten, wie z.B. Web-Clients, E-Government-Basiskomponenten oder anderen Fachanwendungen integriert werden. Dabei muss das Problem gelöst werden, dass durch die Integration von Anwendungen wiederum Abhängigkeiten entstehen. In Kapitel 3 wurden verschiedene Architekturen und Technologien vorgestellt, mit denen diese Abhängigkeiten zwar nicht vollständig beseitigt, aber zumindest gesteuert werden können. Dieser Abschnitt gibt nun Hinweise, wie aus der Vielzahl der möglichen Alternativen, die richtige ausgewählt werden kann.

Bei der Integration von Anwendungen ist hinsichtlich der Plattformunabhängigkeit prinzipiell das gleiche Problem zu lösen wie bei der Realisierung einzelner Fachanwendungen. Die Plattform ist hier jedoch eine andere. Während bei der einzelnen Fachanwendung das Basis-System die Plattform ist (siehe Abschnitt 3.1), ist dies bei der Integration die Gesamtheit der zu integrierenden Anwendungen. In einer integrierten Anwendungslandschaft müssen einzelne Anwendungen zur Erfüllung ihrer Aufgaben Leistungen anderer Anwendungen in Anspruch nehmen. Dadurch entstehen Abhängigkeiten zwischen diesen Anwendungen oder, anders ausgedrückt, Anwendungen werden abhängig von der sie umgebenden Anwendungslandschaft. Diese Abhängigkeiten können im mehrerer Hinsicht problematisch sein:

- Bei der Weiterentwicklung von Anwendungen muss stets darauf geachtet werden, dass diejenigen Anwendungen, die die weiterzuentwickelnde Anwendung nutzen, dies auch weiterhin tun können.
- Der Austausch einer Anwendung durch eine bessere Alternative kann mit hohen Kosten verbunden sein, da die neue Anwendung in das Gesamtsystem integriert werden muss. Das kann beispielsweise dazu führen, dass sich der Wechsel zu einer eigentlich kostengünstigeren Alternative aufgrund zu hoher Integrationskosten nicht rechnet.

- Ändern sich die durch die Anwendungen abgebildeten Prozesse, beispielsweise aufgrund veränderter gesetzlicher Grundlagen, kann die notwendige Anpassung der Anwendungen sehr aufwändig werden, da möglicherweise eine Vielzahl von Anwendungen von der Änderung betroffen ist.

Zwar lassen sich Abhängigkeiten in einer hochintegrierten Anwendungslandschaft nicht vollständig vermeiden, sie können aber durch die Auswahl einer geeigneten IT-Gesamtarchitektur deutlich reduziert werden. Das Grundprinzip ist hier dasselbe wie bei der in Kapitel 3 behandelten Abhängigkeit vom Basis-System: Abhängigkeiten können durch Entkopplung und die Verwendung von Standards reduziert werden.

Entkopplung bedeutet in diesem Zusammenhang, dass die Schnittpunkte zwischen den Anwendungen möglichst dünn gestaltet werden. Je weniger eine Anwendung über eine andere Anwendung, deren Leistung sie nutzen will, wissen muss, umso geringer wird die Abhängigkeit von dieser anderen Anwendung sein. In den folgenden Abschnitten werden verschiedene Mittel zur Kapselung vorgestellt:

- Serviceorientierte und komponentenbasierte Architekturen (siehe Abschnitt 4.1.3.2)
- Einsatz einer vermittelnden Instanz zwischen den Anwendungen (siehe Abschnitte 4.1.3.3, 4.1.3.4 und 4.1.3.5).

Das zweite wichtige Mittel zur Reduktion von Abhängigkeiten ist die Verwendung allgemein anerkannter Standards. Kommunizieren die Anwendungen über standardisierte Schnittstellen[1], können einzelne Anwendungen leichter ausgetauscht werden, da die Auswahl an alternativen Anwendungen, die ebenfalls diesen Standard verwenden, im Allgemeinen größer ist als bei proprietären Protokollen.

Bei der Auswahl einer IT-Gesamtarchitektur spielen viele Faktoren eine Rolle. In den nachfolgenden Abschnitten werden verschiedene dieser Faktoren vorgestellt, meist verbunden mit konkreten Fragen, deren Beantwortung hilfreich bei der Suche nach einer geeigneten Systemarchitektur ist. Bei der Beantwortung dieser Fragen sollten zwei Grundregeln eingehalten werden:

- Erstens sollten sie möglichst beantwortet werden, ohne dass bereits eine bestimmte Wunscharchitektur eine Tendenz vorgibt. Die Gefahr ist sonst groß, Fragen so zu beantworten, dass sich genau die gewünschte Architektur ergibt, obwohl sie möglicherweise gar nicht so ideal ist, wie ursprünglich angenommen.
- Zweitens sollten bereits absehbare zukünftige Änderungen in der IT-Landschaft bei der Beantwortung der Fragen berücksichtigt werden. Bei einigen Fragen, wo dies besonders wichtig ist, ist dies explizit in die Formu-

[1] Siehe SAGA 2.1, Abschnitt 9.3.4.

lierung der Frage eingeflossen. Im Prinzip gilt dieser Hinweis aber für alle Fragen.

Darüber hinaus können natürlich immer nur Tendenzen für eine bestimmte Architektur angegeben werden, da die verschiedenen beteiligten Faktoren in der Praxis häufig mit unterschiedlich starkem Gewicht auftreten.

4.1.3.1 Silo-Architekturen

Silo-Architekturen bieten den Vorteil, dass die Plattformunabhängigkeit jeweils isoliert für die einzelnen Anwendungen betrachtet werden kann. Abhängigkeiten zwischen den Anwendungen gibt es dagegen nicht. Das heißt jedoch nicht, dass Plattformunabhängigkeit dann gar kein Thema mehr ist. Sie ist nur kein Thema mehr hinsichtlich der Integration. Plattformabhängigkeiten bezüglich des Betriebssystems oder des verwendeten Datenbank-Management-Systems müssen natürlich bei der Realisierung der einzelnen Anwendungen weiterhin berücksichtigt werden (siehe Kapitel 3). Es gilt also zu prüfen, ob Silo-Architekturen verwendet werden können.

Die erste Frage, die bei der Entwicklung einer IT-Gesamtarchitektur zu stellen ist, ist die Frage, ob es überhaupt fachliche Abhängigkeiten zwischen den Anwendungen der IT-Landschaft gibt.

Gibt es oder wird es in Zukunft fachliche Abhängigkeiten zwischen den Anwendungen geben?

Bei der Beantwortung dieser Frage können verschiedene Detailfragen hilfreich sein:

Gibt es oder wird es in Zukunft Fachanwendungen geben, die Arbeitsergebnisse anderer Fachanwendungen benötigen?

Wenn man die Fragen mit „ja“ beantwortet die Abhängigkeit offensichtlich und daher klar, dass die Anwendungen integriert werden müssen. Diejenige Fachanwendung, die die Arbeitsergebnisse einer anderen Anwendung benötigt, kann ohne die andere Anwendung oder ein entsprechendes Surrogat nicht betrieben werden.

Manchmal sind fachliche Abhängigkeiten jedoch nicht so offensichtlich.

Gibt es oder wird es in Zukunft Abhängigkeiten zwischen den Daten verschiedener Anwendungen geben?

Wird diese Frage mit "Ja" beantwortet, ist das ein Indiz für eine Abhängigkeit. Folgendes Beispiel verdeutlicht dies:

Beispiel 2 Das Vermessungs- und Katasteramt des Amtsbezirks X hat zwei Fachanwendungen, eine zur Pflege der Liegenschaftskarte und eine zur Pflege der Liegenschaftsbeschreibung. Die Liegenschaftskarte enthält unter anderem Informationen zur Geometrie von Flurstücken und Gebäuden sowie deren exakte Lage (Straße, Hausnummer usw.). Die Liegenschaftsbeschreibung enthält Informationen zu Flurstückbezeichnungen, Flächeninhalten, Eigentümerangaben und amtlichen Bodenschätzungen.

Oberflächlich betrachtet, handelt es sich um disjunkte Informationen. Bei genauerer Prüfung wird jedoch schnell klar, dass hier Abhängigkeiten bestehen. Ändert sich beispielsweise aufgrund einer vorgenommenen Vermessung die Geometrie eines Flurstücks, wird sich auch sein Flächeninhalt ändern (siehe Abbildung 14). Werden nun beide Anwendungen als Monolithen (siehe Abschnitt 3.2.3) realisiert, müssen die Daten beider Anwendungen angepasst werden. Wird dabei eine der beiden Anwendungen vergessen, führt dies zu einer Inkonsistenz, da Geometrie und Flächeninhalt eines Flurstücks nicht mehr zusammen passen.

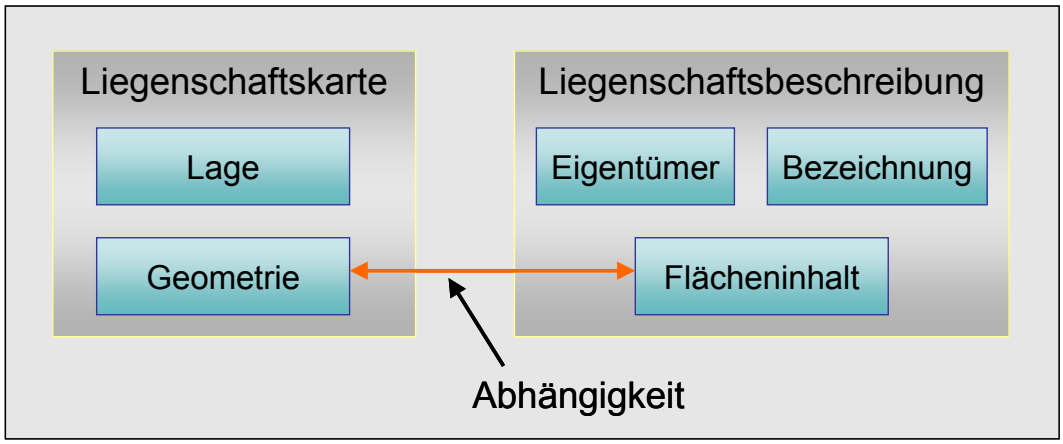


Abbildung 14: Datenabhängigkeit zwischen Geometrie und Flächeninhalt

Gibt es oder wird es in Zukunft fachlich gleichwertige Funktionen geben, die von mehreren Anwendungen ausgeführt werden?

Wird diese Frage mit "Ja" beantwortet, ist das ebenfalls ein Indiz für eine Abhängigkeit. Auch dazu ein Beispiel:

Beispiel 3 Die Behörde A bietet verschiedene Antragsverfahren an, unter anderem eines zur Erteilung eines Fischereischeins und eines zur Erteilung einer Gaststättenerlaubnis. Die im Verfahren zu prüfenden Anforderungen an den Antragsteller sind in den verschiedenen Verfahren weitgehend unterschiedlich. Allen Verfahren gemeinsam ist jedoch, dass der Antragsteller seine Adressdaten angeben muss. Die dem Nutzer zu diesem Zweck zur Verfügung gestellten Eingabemasken sind in allen Fällen identisch. Hier ist die Existenz gleichwertiger Funktionen in den Verfahren somit offensichtlich. Aber selbst wenn die Masken nicht identisch gestaltet sind, folgt daraus noch nicht, dass keine Übereinstimmungen existieren. Bei genauerer Prüfung kann sich beispielsweise herausstellen, dass die für die Überprüfung der Korrektheit der eingegebenen Daten benötigte Funktionalität identisch ist. Auch in diesem Fall gibt es also eine gleichwertige Funktionalität in den Verfahren.

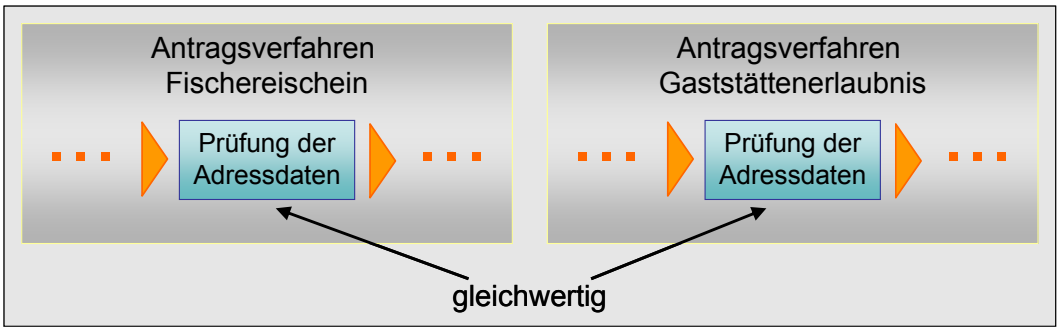


Abbildung 15: Gleichwertige Funktionalität in Teilverfahren

Werden nun die Anwendungen aus Beispiel 3 als Monolithen realisiert, tritt – ähnlich wie beim Beispiel 2 – dann ein Problem auf, wenn sich etwas ändert, in diesem Fall die Regeln, nach denen die Adressdaten überprüft werden.[1] In diesem Fall müssen alle Anwendungen entsprechend angepasst werden, was Aufwendungen verursacht und zusätzlich das Risiko birgt, dass eine der anzupassenden Anwendungen vergessen wird.

Wie die obigen Ausführungen zeigen, muss die Frage, ob fachliche Abhängigkeiten zwischen den Fachanwendungen der IT-Landschaft bestehen, gründlich geprüft werden, da nicht alle Abhängigkeiten sofort erkennbar sind. Aber auch, wenn nach gründlicher Prüfung feststeht, dass keinerlei fachliche Abhängigkeiten bestehen, ist eine Silo-Architektur (siehe Abschnitt 3.2.3) dennoch aus den in Abschnitt 3.2.3.1 genannten Gründen nach Möglichkeit zu vermeiden.

[1] Bekanntlich ist das einzige, was sich nicht ändert, die Tatsache, dass sich alles ändert.

In manchen Fällen sprechen nichttechnische Gründe gegen eine Integration von Anwendungen, obwohl fachliche Abhängigkeiten bestehen. Das könnte beispielsweise dann der Fall sein, wenn bestimmte Daten aus Gründen des Datenschutzes oder aus Geheimschutzgründen nicht übergreifend genutzt werden dürfen. Auch dann ist eine Realisierung als Monolith nicht zweckmäßig. Beispielsweise könnte es möglich sein, Funktionen übergreifend zu nutzen, sofern diese Funktionen nur einen bestimmten Teil der Daten nutzen, aus dem sich z.B. nicht auf die betroffene Person zurück schließen lässt. Hier muss jedoch im Einzelfall genau geprüft werden, ob und welche Integrationen noch möglich sind.

Letztlich sind bei der Entscheidung über die Realisierung der Anwendung die hierzu gegebenen Empfehlungen von SAGA zu beachten (siehe auch Kapitel 6 "Computational Viewpoint: Referenz-Software-Architektur" in SAGA 3.0)."

4.1.3.2 Schnittstellendefinitionen

Werden fachliche Abhängigkeiten identifiziert und sprechen auch keine sonstigen Gründe gegen eine Integration, sollten die Anwendungen integriert werden. Für die Frage, wie dies zu geschehen hat und wie die resultierenden Abhängigkeiten reduziert werden können, bietet sich jedoch ein breites Spektrum an Möglichkeiten, das in diesem und den nachfolgenden Abschnitten untersucht wird.

Grundlegend für die Auswahl einer geeigneten IT-Gesamtarchitektur ist die Frage, ob die zu integrierenden Anwendungen und Komponenten mit Hilfe einer einheitlichen Technologiefamilie implementiert sind.

Müssen jetzt oder in Zukunft heterogene Anwendungen und Komponenten integriert werden, d.h. Anwendungen und Komponenten, die mit Hilfe unterschiedlicher Technologiefamilien (z.B. Java und .NET) implementiert sind?

Die Antwort auf diese Frage gibt Auskunft darüber, welches Maß an Unabhängigkeit zwischen den zu integrierenden Anwendungen und Komponenten untereinander benötigt wird. Anwendungen, die mit Hilfe ein und derselben Technologiefamilie implementiert sind, können anders integriert werden (siehe die Ausführungen zu komponentenbasierten Architekturen am Ende dieses Abschnitts) als solche, bei denen dies nicht der Fall ist.

Wichtig bei der Beantwortung der Frage ist, nicht nur die aktuelle Situation zu berücksichtigen, sondern auch, so weit möglich, zukünftige Situationen. Insbesondere sollte Augenmerk auf die folgende Frage gelegt werden:

Wie stark kann die weitere Entwicklung der zu integrierenden Anwendungen und Komponenten beeinflusst werden?

Selbst wenn die zu integrierenden Anwendungen gegenwärtig mit Hilfe der gleichen Technologiefamilie implementiert sind, heißt das noch nicht, dass dies auch zukünftig so bleibt. Bestehen keine oder nur geringe Einflussmöglichkeiten auf die weitere Entwicklung dieser Anwendungen, sollte man auf etwaige Änderungen vorbereitet sein. Somit sind geringe Einflussmöglichkeiten ein starkes Indiz für ein "Ja" auf die Frage, ob jetzt oder in Zukunft heterogene Anwendungen und Komponenten integriert werden.

Klar ist, dass die weitere Entwicklung von Anwendungen stark beeinflusst werden kann, wenn es sich um Eigen- oder Auftragsentwicklungen handelt. Beim Kauf kann das dagegen schon anders aussehen. Möglicherweise gibt es für eine bestimmte Fachanwendung nur ein geeignetes Produkt und das ist unglücklicherweise mit Hilfe der "falschen" Technologiefamilie implementiert worden.

Noch geringer sind die Einflussmöglichkeiten in der Regel dann, wenn Anwendungen integriert werden müssen, die unter der Kontrolle einer anderen Organisation (Behörde oder Unternehmen) stehen.[1]

Gibt es Schnittstellen nach außen, d.h., werden Dienste anderer Organisationen genutzt oder Dienste für andere Organisationen bereit gestellt, so dass eine Integration mit diesen sinnvoll ist?

Werden Verfahren abgebildet, an denen noch andere Organisationen aktiv beteiligt sind, muss es Schnittstellen nach außen geben.

Beispiel 4 Ein Antragsverfahren für einen Gewerbeschein, das so oder so ähnlich ablaufen könnte. Federführend ist das Gewerbeamt einer Gemeinde. Das Gewerbeamt nimmt den Antrag zunächst entgegen, prüft ihn auf formale Korrektheit und führt erste inhaltliche Prüfungen durch. Anschließend wird der Antrag an das Landratsamt übergeben und dort weiter bearbeitet. Nach Fertigstellung der Bearbeitung durch das Landratsamt wird die Bearbeitung durch das Gewerbeamt fortgesetzt.

[1] Das gilt gleichermaßen, wenn innerhalb einer Behörde verschiedene Abteilungen existieren, die jeweils über eine eigene autarke IT-Landschaft verfügen. Unter den hier betrachteten Gesichtspunkt ist in diesem Fall eine abteilungsübergreifende Integration einer behördenübergreifenden Integration gleichzusetzen.

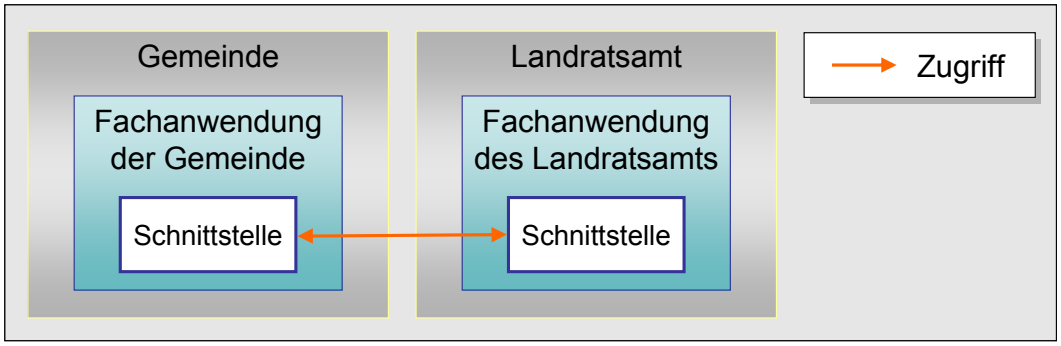


Abbildung 16: Behördenübergreifendes Antragsverfahren

Hier ist offensichtlich, dass es Schnittstellen zwischen der Fachanwendung des Gewerbeamtes und einer Fachanwendung des Landratsamtes geben muss, sofern die Weiterleitung des Antrags an das Landratsamt und die Zurückgabe an das Gewerbeamt medienbruchfrei geschehen sollen.

Ebenso klar ist, dass es Schnittstellen nach außen geben muss, wenn für die Abarbeitung eines Verfahrens Informationen benötigt werden, die nur einer anderen Behörde bekannt sind. Beispielsweise ist denkbar, dass die Behörde A verschiedene Antragsverfahren anbietet, bei denen die Korrektheit der vom Antragsteller angegebenen Adresse bei der zuständigen Meldebehörde erfragt wird. In diesem Fall obliegt es der Meldebehörde, einen entsprechenden Dienst bereit zu stellen, während Behörde A ihre Fachanwendungen so integrieren muss, dass sie den Dienst nutzen können.

Behörden, wie z.B. das Bundesverwaltungsamt (BVA) oder kommunale Rechenzentren, deren Aufgabe unter anderem darin besteht, IT-Dienstleistungen für andere Behörden bereit zu stellen, bieten i.d.R. ebenfalls Schnittstellen in ihren Anwendungen nach außen zu anderen Behörden an. Aus dem Blickwinkel der Integration sind hier allerdings nur solche Dienstleistungen interessant, die von Anwendungen genutzt werden, in Abgrenzung zu Dienstleistungen, die von Mitarbeitern genutzt werden.

Beispiel 5 Das vom BVA betriebene Travel Management System (TMS) ist eine Anwendung, die von Mitarbeitern verschiedener Behörden über eine Web-Schnittstelle genutzt werden kann. Dieser Dienst wird also nicht von anderen Anwendungen genutzt, sondern von Personen. Als Web-Anwendung ist das TMS somit unter Integrationsgesichtspunkten nicht interessant. Andererseits werden im TMS Identitäten verwaltet, die auch in anderen Behörden, beispielsweise dem BMI, in den unterschiedlichsten Anwendungen genutzt und verwaltet werden. Die Verwaltung dieser Identitäten als Dienst für andere Behörden könnte daher unter Integrationsgesichtspunkten durchaus interessant sein.

Schwieriger ist die Frage zu beantworten, ob externe Dienste genutzt werden sollen, wenn es keine fachlichen Erfordernisse gibt, die dies erzwingen. Die Nutzung externer Dienste führt dazu, dass bei der Entwicklung der IT-Gesamtarchitektur Anwendungen oder Komponenten berücksichtigt werden müssen, die nicht unter der eigenen Kontrolle stehen und deren Entwicklungszyklen nicht oder nur bedingt beeinflusst werden können. Trotzdem ist es meist empfehlenswert, solche Dienste zu nutzen. Die Entwicklung oder Beschaffung sowie die Pflege eigener Anwendungen, die diese Dienste bereit stellen, ist i.d.R. mit Aufwendungen und Kosten verbunden, die durch das Mehr an gewonnener Kontrolle nicht ausgeglichen werden.

Müssen Anwendungen und Komponenten integriert werden, die nicht in einer einheitlichen Technologie implementiert wurden oder für die dies nicht auf Dauer garantiert werden kann, bietet sich eine serviceorientierte Architektur (SOA, siehe Abschnitt 3.2.3) an. Kennzeichnend für eine SOA ist, dass die Fachlogik durch eine Reihe von lose gekoppelten, unabhängigen Services realisiert wird. Als Architekturparadigma ist SOA zunächst unabhängig von konkreten Technologien. Die übliche Technologiefamilie für die Umsetzung einer SOA ist jedoch die Web-Service-Technologiefamilie. Deren Bestandteile SOAP, WSDL und XSD werden in SAGA 3.0 referenziert[1].

Für die Integration von Anwendungen, die mit Hilfe unterschiedlicher Technologiefamilien implementiert wurden, ist eine auf Web-Services basierende serviceorientierte Architektur besonders gut geeignet, weil Standardprotokolle (wie z.B. SOAP) und Standarddatenformate (wie z.B. XML) verwendet werden. Dadurch können Technologieinkompatibilitäten überbrückt werden. Beispielsweise kann eine mit Java entwickelte Anwendung einen Dienst nutzen, der mit .NET implementiert wurde.

Die wichtigste Aufgabe beim Entwurf einer serviceorientierten Architektur ist die Definition geeigneter Schnittstellen. Hier ist folgendes zu beachten:

1. Die Kommunikation mit einem Service ist vergleichsweise langsam. Deshalb sollte aus Performanzgründen die Anzahl der benötigten Aufrufe nicht zu groß werden.
   - Services werden über eine Netzwerkverbindung angesprochen. Daher ist die Kommunikation mit einem Service mit einer gewissen Latenzzeit verbunden.
   - Für die mittels Services realisierten Behördenprozesse gelten häufig hohe Sicherheitsanforderungen, u.a. weil häufig mit personenbezogenen Daten gearbeitet wird. Es müssen deshalb Sicherheitsmechanis-

[1] Siehe SAGA 3.0, Abschnitt 8.6.1

men (Datenverschlüsselung, Authentifizierung, Autorisierung) in die Kommunikation eingebaut werden.

- XML als universelles Datenformat gestattet es, beliebige Nutzdaten in strukturierter Form zu versenden. Allerdings verwenden die beteiligten Anwendungen intern häufig ein anwendungsspezifisches, kompakteres Format. Für den Datenaustausch müssen die Daten in diesen Fällen umgewandelt werden.

2. HTTP und SOAP sind so genannte zustandslose Protokolle, d.h. nach einer Anfrage und einer Antwort ist der Dialog zwischen Client und Service prinzipiell beendet. Ein zustandsbehaftetes Protokoll in diesem Sinne würde weitere Anfrage-/Antwort-Paare vorsehen, von denen jedes inhaltlich auf die Ergebnisse der vorausgegangenen aufbauen könnte, also auf den bereits erreichten Zustand der Verbindung. Eine zustandsorientierte Kommunikation auf der Basis zustandsloser Protokolle ist selbstverständlich möglich; da die Verwaltung des Zustandes jedoch nicht durch das Protokoll geschieht, muss sie durch die Anwendungslogik geleistet werden, und trägt so zu deren Komplexität bei.

Die Anzahl der Aufrufe lässt sich durch die Verwendung von grobgranularen Schnittstellen erreichen. Grobgranulare Schnittstellen sind Schnittstellen, die vergleichsweise viel Funktionalität auf einmal anbieten, im Gegensatz zu feingranularen Schnittstellen, die jeweils eine sehr spezifische Detailaufgabe übernehmen. Grobgranulare Schnittstellen reduzieren somit die Anzahl der benötigten Aufrufe.

Beispielsweise sollte die Überprüfung der Korrektheit von Adressdaten eines Antrags, sofern sie als Service realisiert wird, in einem Stück erfolgen. Dem Service müssen dann beim Aufruf die vollständigen Adressdaten des Antrags sowie sämtliche weiteren Informationen übergeben werden, die für die Prüfung benötigt werden könnten. Für die Prüfung der Korrektheit ist dann genau ein Aufruf erforderlich. Eine feingranulare Realisierung sähe dagegen so aus, dass jeweils einzelne Adressteile, wie Straße, Postleitzahl oder Hausnummer, überprüft würden. Abgesehen davon, dass die Korrektheit einiger dieser Teile nur im Zusammenspiel mit anderen Teilen überprüft werden kann, beispielsweise Straße und Hausnummer, würde diese Form der Realisierung die Anzahl der benötigten Service-Aufrufe deutlich steigern, was aus den oben genannten Gründen nicht zu empfehlen ist.

Will man der Eigenschaft von HTTP und SOAP gerecht werden, dass es sich um zustandslose Protokolle handelt, dann sollten Services kontextfreie Funktionalitäten anbieten. Kontextfreie Funktionalitäten müssen keinen vorausgehenden oder nachfolgenden Zustand berücksichtigen bzw. können ausgeführt werden ohne dass ein bestimmter Zustand zuvor erreicht sein muss. Wenn in dem vorgenannten Beispiel ein Teil der Korrektheitsprüfung die Prüfung der Übereinstimmung von Postleitzahl und Wohnort ist, dann wäre dies bei einer grobgranularen Schnittstelle kontextfrei zu realisieren, da alle Informationen die benötigt werden

(Ort und Postleitzahl) verfügbar wären und bei einer feingranularen Schnittstelle nur kontextsensitiv, da die Funktionalität auf die Ergebnisse der Einzelprüfung von Postleitzahl und Ort zurückgreifen muss. Das heißt, Postleitzahl und Ort müssen irgendwo zwischengespeichert werden, da HTTP und SOAP sich dies als zustandslose Protokolle nicht merken.

Somit gehen die beiden Forderungen nach grobgranulare Schnittstellen und kontextfreie Funktionalitäten Hand in Hand und es lässt sich folgende Regel für die Definition von Service-Schnittstellen aufstellen:

> *Services sollten grobgranulare Schnittstellen und kontextfreie Funktionalitäten anbieten.*

Serviceorientierte Architekturen regeln vor allem die Integration im Großen. Sie machen dabei keine Vorgaben, wie die verschiedenen Services intern umgesetzt werden. Das ist einer der Vorzüge serviceorientierter Architekturen, da sie somit für die Integration von Anwendungen und Komponenten geeignet sind, die unter Verwendung unterschiedlichster Technologien implementiert wurden.

Für die Integration im Kleinen sind serviceorientierte Architekturen jedoch weniger geeignet. Das liegt vor allem daran, dass Services grobgranulare Schnittstellen benötigen (siehe oben). Hier eigenen sich komponentenbasierte Architekturen (siehe Abschnitt 3.2.3) besser. Das gilt sowohl dann, wenn es um die Realisierung eines einzelnen Services geht, als auch dann, wenn es sich um eine vollständige Fachanwendung handelt, die entweder gar nicht mit anderen Anwendungen und Komponenten interagieren muss oder nur mit solchen, die mit Hilfe der gleichen Technologiefamilie implementiert wurden und deren weitere Entwicklung entsprechend beeinflusst werden kann.

Komponentenbasierte Architekturen basieren zumeist auf einer konkreten Komponenten-Technologie, wie z.B. Java oder .NET. Die Realisierung eines Services oder einer Fachanwendung im Wege einer komponentenbasierten Architektur führt daher zu einer Abhängigkeit von einer konkreten Technologieplattform. Diese Abhängigkeit ist jedoch nur dann problematisch, wenn sie der weiteren Entwicklung der IT-Landschaft im Wege steht. Letzteres ist dann der Fall, wenn Anwendungen integriert werden müssen, die mit Hilfe unterschiedlicher Technologiefamilien implementiert wurden (siehe zu Beginn des Kapitels).

Müssen solche Anwendungen jedoch nicht integriert werden, bieten komponentenbasierte Architekturen durchaus ihre Vorteile. Wie oben dargelegt wurde, ist die Kommunikation mit einem Service vergleichsweise langsam. Ferner sind zustandsbehaftete Interaktionen mit einem Web-Service auf Basis von SOAP standardmäßig nicht möglich. Für den Zugriff auf Komponenten gilt das jedoch nicht. Wie das Beispiel der "Stateful Session Beans" aus der Enterprise Java Beans

Technologie[1] zeigt, sind zustandsbehaftete Kommunikationen mit Komponenten durchaus möglich. Auch hinsichtlich der Performance weisen Komponenten häufig Vorteile gegenüber Services auf. Daher können die Schnittstellen für Komponenten feingranularer gewählt werden als die für Services.

Wie bereits angedeutet, darf der Rahmen innerhalb dessen komponentenbasiert integriert wird, nicht zu groß gewählt werden, denn auch hier ist wichtig zu berücksichtigen, welche Einflussmöglichkeiten jetzt und in Zukunft auf die Weiterentwicklung der beteiligten Anwendungen bestehen:

Besteht die Möglichkeit, dass für die Umsetzung des Services jetzt oder zukünftig Komponenten benötigt werden, bei denen kein Einfluss auf die verwendete Implementierungstechnologie genommen werden kann?

Wenn diese Frage mit "Ja" beantwortet wird, sollte der Service nicht komponentenbasiert umgesetzt werden. Statt dessen ist es empfehlenswerter, den Service in Teilservices zu zerlegen und die Teilservices serviceorientiert zu integrieren. Die Umsetzung der einzelnen Teilservices kann dann komponentenbasiert erfolgen.

Hinweise zur Definition von Services und Komponenten

- Services sollten mit Standard-Technologien umgesetzt werden. Das betrifft das Kommunikationsprotokoll (SOAP), das Transportprotokoll (HTTP), die verwendeten Datenformate (XML) und die Spezifikation der Schnittstellen (WSDL, XSD).
- Die bei der Spezifikation von Service-Schnittstellen verwendeten XML-Schemata sollten an einer zentralen Stelle koordiniert werden.
- Bei der Spezifikation der Service-Schnittstellen sollte auf die richtige Granularität der Schnittstellen geachtet werden. Jeder Service muss eine kontextfreie Funktionalität anbieten.
- Für die Implementierung einzelner Services bieten sich komponentenbasierte Architekturen an[2].

[1] Die Enterprise Java Beans Technologie (abgekürzt EJB) ist ein Teil der J2EE-Technologiefamilie.

[2] Das gilt auch dann, wenn es sich um die Umsetzung einer vollständigen Fachanwendung handelt, die entweder gar nicht mit anderen Anwendungen und Komponenten interagieren muss oder nur mit solchen, die zur IT-Landschaft derselben Behörde gehören und mit Hilfe der gleichen Technologiefamilie implementiert wurden.

- Als Komponententechnologie ist Java (J2SE/J2EE) gemäß SAGA obligatorisch, während .NET den Status "unter Beobachtung" hat[1].
- Besteht die Möglichkeit, dass für die Umsetzung des Services jetzt oder zukünftig Komponenten benötigt werden, bei denen kein Einfluss auf die verwendete Implementierungstechnologie genommen werden kann, sollte der Service in geeignete Teilservices zerlegt werden.

4.1.3.3 Einsatz zentraler Integrationssoftware

Die Verwendung von serviceorientierten und komponentenbasierten Architekturen sagt noch nichts darüber aus, ob die integrierten Anwendungen direkt miteinander kommunizieren oder ob eine zentrale Integrationssoftware als Vermittler und Koordinator eingesetzt wird. Für die Frage, ob eine solche Integrationssoftware eingesetzt werden sollte, ist vor allem die Anzahl der zu integrierenden Anwendungen von Bedeutung.

Wie viele Fachanwendungen enthält die IT-Landschaft?

Enthält die IT-Landschaft nur eine Handvoll Fachanwendungen, können diese durch Punkt-zu-Punkt-Integrationen (P2P-Integrationen, siehe Abschnitt 3.2.3) ohne Verwendung einer zentralen Integrationssoftware miteinander verbunden werden. Solche P2P-Integrationen lassen sich i.d.R. vergleichsweise schnell realisieren, führen jedoch, wenn sie im großen Stil verwendet werden, zu so genannte Stovepipe-Architekturen (siehe Abschnitt 3.3.1). Daher sollten sie nur dann eingesetzt werden, wenn nur sehr wenige Anwendungen zu integrieren sind, wobei hier die Berücksichtigung der absehbaren weiteren Entwicklung der IT-Landschaft besonders wichtig ist. Zeichnet sich bereits ab, dass eine derzeit noch kleine IT-Landschaft in absehbarer Zeit um weitere Anwendungen wachsen wird, sollte auf P2P-Integrationen verzichtet werden.

Kommen P2P-Integrationen aus den genannten Gründen nicht in Betracht, sollte eine zentrale Integrationssoftware zur Koordinierung der Integrationen eingesetzt werden.

Zur Auswahl stehen hier

- DOT-Middleware (siehe Abschnitt 4.1.3.5),
- Message-oriented Middleware (siehe Abschnitt 4.1.3.5) oder
- Enterprise Services Buses (siehe ebenfalls Abschnitt 4.1.3.5).

[1] Zur Bedeutung der Klassifizierung dieser Technologien siehe Kapitel „Klassifizierung und Lebenszyklen von Standards“ in SAGA 3.0 (www.kbst.bund.de)

Die Entscheidung für eine Integrationssoftware ist keine Entweder-Oder-Entscheidung. Vielmehr lassen sich die verschiedenen Typen von Integrationssoftware kombinieren. In den nachfolgenden Abschnitten wird an den entsprechenden Stellen darauf hingewiesen.

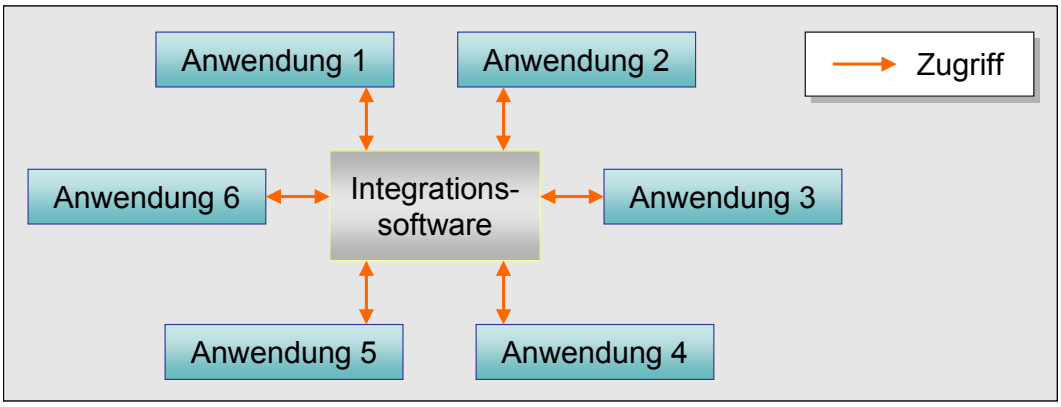


Abbildung 17: Vermeidung von PTP-Verbindungen durch Integrationssoftware

Durch den Einsatz einer zentralen Integrationssoftware werden Abhängigkeiten zwischen den einzelnen Anwendungen der IT-Landschaft reduziert. Es ergibt sich aber eine Abhängigkeit hinsichtlich des konkret ausgewählten Integrationsprodukts. Das konkrete Maß dieser Abhängigkeit kann jedoch stark variieren und durch geeignete Maßnahmen auf ein notwendiges Minimum reduziert werden.

Das wichtigste Mittel zur Reduzierung dieser Abhängigkeit ist die Verwendung von Integrationsprodukten und Anwendungen, die allgemein anerkannten Standards folgen. Dadurch wird die Weiterentwicklung der IT-Landschaft in zweifacher Hinsicht unterstützt:

- Standardkonforme Integrationssoftware kann leichter durch andere standardkonforme Integrationssoftware ersetzt werden, da die grundlegenden Integrationsfragen, wie die Art der Kommunikation mit den Anwendungen und die verwendeten Datenformate, einheitlich geregelt sind.
- Aus dem gleichen Grund können die durch die Integrationssoftware gesteuerten Anwendungen leichter durch andere standardkonforme Anwendungen ersetzt werden.

Standards helfen jedoch nur dann, wenn sie auch eingehalten werden. Die Verwendung von standardkonformer Software sollte daher durch organisatorische Maßnahmen flankiert werden. Viele Produkte, auch solche die standardkonform sind, bieten neben den Standardfunktionen weitere Funktionen an. Die Nutzung dieser proprietären Funktionen führt jedoch zu einer Produktabhängigkeit, die die Weiterentwicklung der IT-Landschaft erheblich erschweren kann. Daher sollten proprietäre Funktionen nur dann genutzt werden, wenn dies aus fachlichen Gründen unvermeidbar ist. Sofern es keine Möglichkeit gibt, die Verwendung

proprietärer Funktionen technisch zu verhindern, beispielsweise durch Ausschalten dieser Funktionen mittels entsprechender Konfiguration, muss sie durch organisatorische Anweisungen verhindert werden.

Ebenfalls von Bedeutung ist die Frage, wie aufwändig die Administration einer Integrationssoftware zu erlernen ist. Je höher die zu überwindende Lernkurve ist, desto mehr entsteht eine, zumindest psychologische, Bindung an das konkrete Produkt, da sich die getätigten Lernaufwendungen ja lohnen sollen. Bei der Beschaffung einer Integrationssoftware sollte daher darauf geachtet werden, dass die Administration der Software möglichst leicht zu erlernen ist.

Obwohl eine Integrationssoftware keine Fachanwendung ist, gelten hinsichtlich der Abhängigkeit vom Betriebssystem oder einem Datenbank-Management-System für Integrationssoftware die gleichen Regeln wie für Fachanwendungen. Neben der Standardkonformität müssen bei der Auswahl einer Integrationssoftware also auch andere Plattformabhängigkeiten berücksichtigt werden, beispielsweise bezüglich des Betriebssystems oder des verwendeten DBMS.

Hinweise für die Beschaffung einer Integrationssoftware:

- Die verwendete Integrationssoftware sollte allgemein anerkannten Standards folgen.
- Sie sollte so wenig wie möglich an proprietären Funktionen beinhalten bzw. diese sollten möglichst nicht genutzt werden.
- Die Integrationssoftware sollte weder von einem bestimmten Betriebssystem noch von einem bestimmten Datenbank-Management-System abhängig sein.
- Die Administration der Integrationssoftware sollte möglichst leicht zu erlernen sein.

4.1.3.4 Integrationssoftware in serviceorientierten Architekturen

Wie bereits in Kapitel 3 dargelegt, sind Services für sich allein genommen noch nicht besonders leistungsfähig. Erst durch das Zusammenspiel mehrerer Services können die Prozesse einer Behörde vollständig abgebildet werden. Zu diesem Zweck müssen die beteiligten Services und ihr Zusammenwirken optimal aufeinander abgestimmt werden. Man spricht daher auch oft vom Orchestrieren der Services.

Beispiel 6 Die vereinfachte Ablaufbeschreibung eines Antragsverfahrens für einen Gewerbeschein, das so oder so ähnlich aussehen könnte (siehe auch Beispiel 4).

1. Nach Eingang eines Antrags ruft die zuständige Fachanwendung des Gewerbeamts zunächst einen Service auf, der die formale Korrektheit des Antrags prüft.
2. Anschließend bereitet sie den Antrag für die weitere Bearbeitung durch einen Sachbearbeiter auf.
3. Nachdem der Sachbearbeiter die erforderlichen Schritte vorgenommen hat, wird der Antrag von der Fachanwendung an eine entsprechende Fachanwendung des Landratsamtes weitergeleitet.
4. Nach weiterer Bearbeitung durch einen Sachbearbeiter des Landratsamtes schickt die Fachanwendung des Landratsamtes den Antrag zurück an die Fachanwendung des Gewerbeamtes.
5. Ferner schickt die Fachanwendung des Landratsamtes das Ergebnis der Prüfung durch das Landratsamt an verschiedene weitere Organisationen (Weitermeldungsempfänger) zur Kenntnisnahme, unter anderem an das Finanzamt.
6. Eine Fachanwendung des Finanzamts weist dem Antragsteller daraufhin eine Steuernummer zu.
7. Nach Rückerhalt des Antrags durch die Fachanwendung des Landratsamts weist die Fachanwendung des Gewerbeamts den Antrag dem zuständigen Sachbearbeiter zur abschließenden Bearbeitung zu.
8. Ist die abschließende Bearbeitung erfolgt, schickt die Fachanwendung einen Bescheid an den Antragsteller per Email. In der Kommunikation mit dem Antragsteller nutzt die Fachanwendung die BundOnline-2005-Basiskomponente[1] "Virtuelle Poststelle".[2]

Obwohl das Beispiel längst nicht alle Details eines solchen Verfahrens enthält (beispielsweise geht es davon aus, dass alle Prüfungen positiv ausgehen) zeigt es bereits, dass bei der IT-gestützten Abwicklung des Verfahrens eine Orchestrierung der beteiligten Anwendungen und Services erforderlich ist. Diese Orchestrierung kann auf verschiedene Art und Weise erfolgen.

In der einfachsten Variante orchestrieren sich die Fachanwendungen selbst. Die für die Zusammenarbeit notwendige Integrationslogik ist dann in den Fachan-

[1] In SAGA 3.0 wird die BundOnline-2005-Basiskomponente, „Virtuelle Poststelle“, als EfA-System bezeichnet.

[2] Siehe www.kbst.bund.de, SAGA 3.0 im Anhang (A.4).

wendungen enthalten, d.h., eine Fachanwendung weiß zu jedem Zeitpunkt selbst, welchen Service sie als nächstes benötigt. Sie lokalisiert dann den benötigten Service mit Hilfe eines Service Brokers (siehe Abschnitt 3.2.3) und ruft anschließend den Service auf.

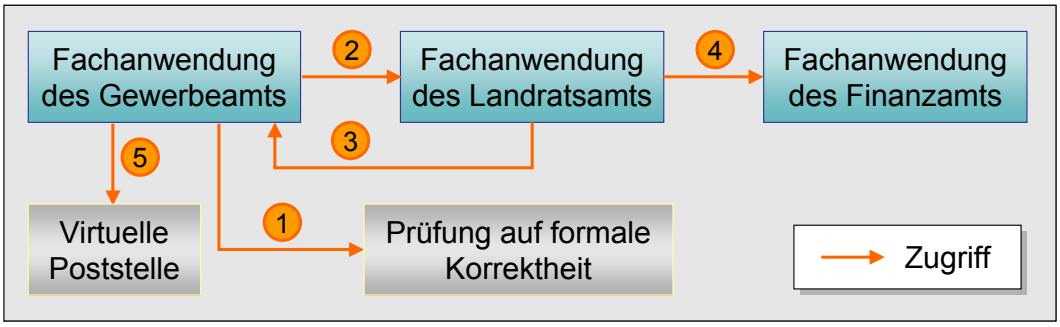


Abbildung 18: Antragsverfahren mit selbstorchestrierenden Fachanwendungen

Alternativ kann die Integrationslogik aus den Fachanwendungen heraus gezogen und in eine zentrale Integrationssoftware, einen so genannten Enterprise Service Bus ausgelagert werden.

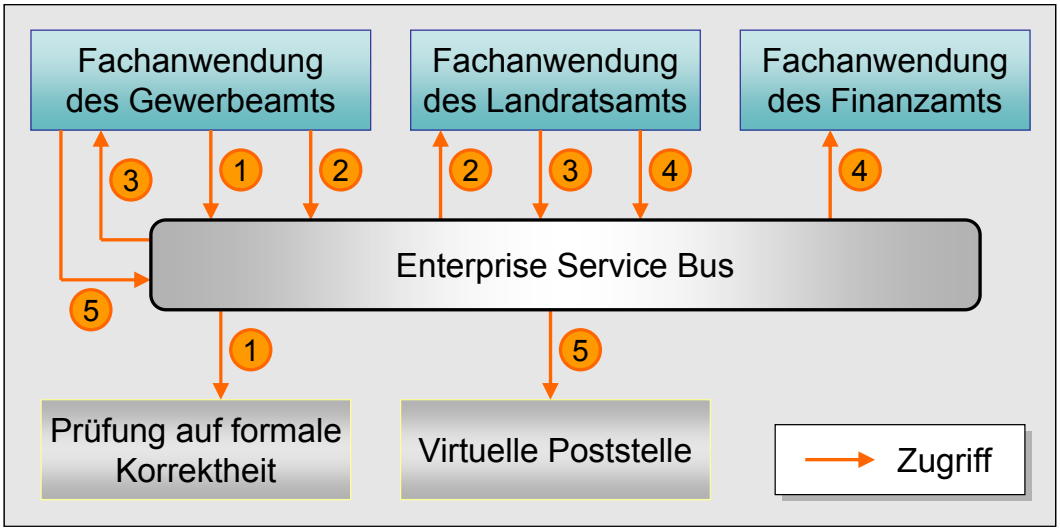


Abbildung 19: Antragsverfahren mit Enterprise Service Bus

Ein *Enterprise Service Bus* (ESB) ist eine Integrationssoftware für die Umsetzung einer serviceorientierten Architektur, die aus folgenden Bestandteilen besteht:

- nachrichtenorientierter Middleware
- content-based Router
- Datentransformer

Eine **nachrichtenorientierte Middleware** (Message-oriented Middleware, MOM)[1] ist für die Organisation von Nachrichten zuständig. Typische Aufgaben einer MOM sind:

- *Entgegennehmen und Ausliefern von Nachrichten*
  Dazu gehört insbesondere das Lokalisieren des Empfängers.
- *Zwischenspeichern von Nachrichten*
  Eine MOM kann eingehende Nachrichten speichern und protokollieren, welche Nachrichten ausgeliefert wurden. Ist der Empfänger zeitweilig nicht erreichbar, werden die Zustellungsversuche so lange wiederholt, bis der Empfänger wieder erreichbar ist (*Guaranteed Delivery*).
- *Auswählen von Empfängern*
  Dazu wird in der MOM ein so genanntes *Topic* (Thema) eingerichtet, beispielsweise das Topic "Eingang eines Antragsformulars". Anwendungen und Komponenten, die über Ereignisse informiert werden sollten, die zu diesem Topic gehören, können dann bei der MOM registriert werden und erhalten alle Nachrichten, die zu dem entsprechenden Topic eingehen.
- *Priorisierung von Nachrichten*
  Manche Nachrichten sind wichtiger als andere und müssen daher schneller bearbeitet werden. Eine MOM kann mittels Priorisierung dafür sorgen, dass Nachrichten mit hoher Priorität Nachrichten mit niedrigerer Priorität überholen. Die wichtigeren Nachrichten werden also früher ausgeliefert, obwohl sie später abgeschickt wurden.

Ein **content-based Router** ist ein Nachrichtenverteiler, der in der Lage ist, die Empfänger einer Nachricht basierend auf dem Inhalt der Nachricht zu ermitteln. Mit Hilfe eines content-based Routers kann der Prozess der Nachrichtenverteilung weiter automatisiert und dadurch die Abhängigkeit zwischen den Kommunikationsbeteiligten weiter reduziert werden.

Ein **Datentransformer** ist dafür zuständig, die Daten einer Nachricht so zu transformieren, dass der Empfänger sie versteht. Auch Datentransformer dienen dem Zweck, die Abhängigkeiten zwischen den Kommunikationsbeteiligten zu reduzieren. Der Sender einer Nachricht kann die zur Nachricht gehörenden Daten in einem ihm genehmen Format versenden, ohne dabei Rücksicht auf den Empfänger nehmen zu müssen. Sind die Daten in einem Format abgelegt, das der Empfänger nicht versteht, werden sie zuerst durch einen Datentransformer in ein geeignetes Format umgewandelt.

[1] Nicht zu verwechseln mit dem Microsoft Operation Manager, der auch mit MOM abgekürzt wird.

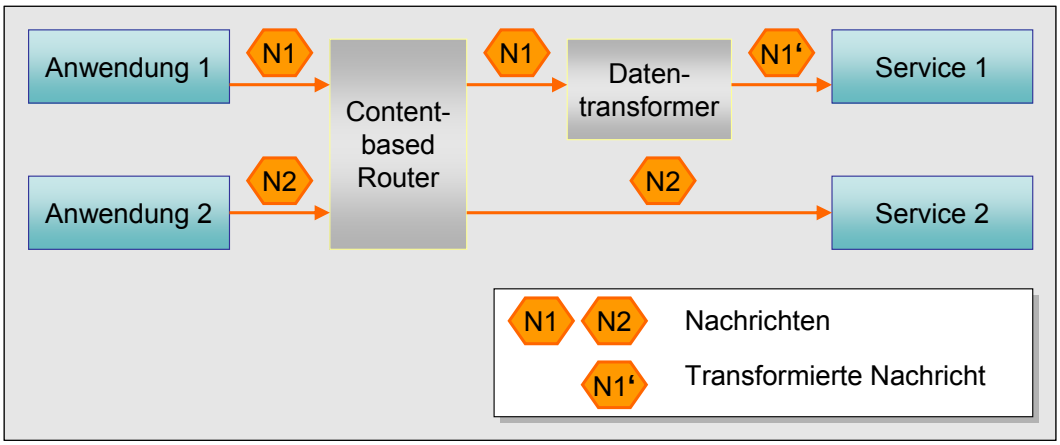


Abbildung 20: Content-based Routing und Datentransformation

Im Folgenden werden Hinweise gegeben, wann der Einsatz eines ESB in Betracht gezogen werden sollte.

Werden änderungsanfällige Prozesse umgesetzt?

Ist die Integrationslogik in den Fachanwendungen implementiert, führt dies zu einer hohen Abhängigkeit der Anwendungen von den Services, die sie nutzen. Ändern sich beispielsweise die Regeln des abgebildeten Verfahrens dahingehend, dass ein zusätzlicher Service benötigt wird, muss die Fachanwendung angepasst werden.

Beispiel 7 Im Antragsverfahren aus Beispiel 6 für einen Gewerbeschein könnten z.B. aufgrund geänderter gesetzlicher Vorschriften weitere Weitermeldungsempfänger hinzukommen, die über das Ergebnis der Antragsprüfung durch das Landratsamt informiert werden müssen. Beispielweise könnte die Verpflichtung zur Anmeldung beim Handelsregistergericht vom Antragsteller an das Landratsamt übergehen. In diesem Fall müsste die Fachanwendung des Landratsamts so angepasst werden, dass sie auch das Handelsregistergericht informiert.

Denkbar ist auch, dass Anwendungen zusammengelegt werden.

Beispiel 8 Beispielsweise könnte vom Landratsamt ein Service bereit gestellt werden, der die formale Prüfung auf Korrektheit eines Antrags übernimmt. Die entsprechenden Services der einzelnen Gewerbeämter werden dann obsolet und die Fachanwendungen der Gewerbeämter müssten so angepasst werden, dass sie zukünftig auf den vom Landratsamt bereit gestellten Service zugreifen.

Wird hingegen ein ESB eingesetzt, wird die Modellierung der Geschäftsprozesse in der ESB-Software durchgeführt und dadurch die Abhängigkeit der Anwendungen von den Services, die sie nutzen, reduziert. Bei einer Änderung der Verfahrensregeln muss nur die Modellierung im ESB angepasst werden. Implementierungsaufwendungen durch Anpassung der Fachanwendungen entstehen nicht.

Müssen Services genutzt werden, die lange oder nicht vorhersagbare Antwortzeiten haben?

Nutzt eine Fachanwendung einen Service, der eine lange oder nicht vorhersagbare Antwortzeit hat, kann die Anwendung nicht warten, bis der Service seine Arbeit verrichtet hat.

Beispiel 9 Ein Verfahren für die Beantragung eines Führerscheins wird als Online-Dienstleistung Bürgern über eine Web-Applikation zur Verfügung gestellt. Um einen Antrag zu stellen, füllt ein Bürger ein Web-Formular aus und schickt das ausgefüllte Formular per Mausklick ab. Die Web-Anwendung muss nun das Formular in einem Dokumenten-Management-System (DMS) ablegen und an eine Fachanwendung für die Bearbeitung des Antrags senden.

Es darf erwartet werden, dass die Ablage im DMS einigermaßen schnell von Statten geht. Die Fachanwendung wird dagegen den Antrag nicht eigenständig bearbeiten können. Vielmehr wird dazu die Interaktion mit einem Sachbearbeiter erforderlich sein. Wann dies jedoch geschieht, ist völlig unklar. Darüber hinaus wird die Bearbeitung des Antrags einige Zeit in Anspruch nehmen, da sie möglicherweise die Kommunikation mit weiteren Anwendungen oder Behörden beinhaltet. Es wäre also absurd, die Web-Applikation so lange warten zu lassen, bis die Bearbeitung des Antrags abgeschlossen ist.

Statt dessen wird die Web-Applikation beispielsweise so vorgehen, dass sie eine Nachricht erzeugt, das ausgefüllte Formular an die Nachricht anhängt und die Nachricht sowohl an die Fachanwendung als auch an das DMS schickt. Anschließend teilt sie dem Nutzer mit, dass der Antrag weitergereicht wurde und dass er über den Ausgang des Verfahrens per Email informiert wird.

Die im Beispiel beschriebene Vorgehensweise realisiert eine ereignisgetriebene Architektur. Die Web-Anwendung reagiert hier auf das Ereignis Formulareingang, indem sie eine Nachricht erzeugt und auf diese Weise weitere Komponenten (Fachanwendung, DMS) über das Ereignis informiert. Hier bietet sich der Einsatz eines ESB[1] an, der für die Verteilung der Nachricht an die Empfänger zuständig ist.

Sind Anwendungen oder Komponenten beteiligt, deren permanente Verfügbarkeit nicht garantiert ist?

Beispiel 10 Die Behörde aus Beispiel 9 möchte ihre Dienstleistung gerne rund um die Uhr, d.h. im 24/7-Betrieb anbieten. Allerdings ist sie gelegentlich gezwungen die Fachanwendung oder das DMS zu Wartungszwecken herunterzufahren. Es wäre äußerst unschön, wenn die Web-Anwendung in diesem Fall einem Bürger, der gerade mit Mühe ein Formular ausgefüllt hat, mitteilen würde, dass diese Mühe umsonst war, weil die benötigte Fachanwendung nicht erreichbar ist.

Wird eine MOM verwendet, kann dieses Problem leicht gelöst werden. Eine MOM kann Nachrichten zwischenspeichern und Zustellungsversuche wiederholen. Somit wird sichergestellt, dass die Nachricht jeden Empfänger genau einmal erreicht, und zwar auch dann, wenn einer der Empfänger vorübergehend nicht erreichbar ist[2].

Natürlich kann diese Leistung auch von den Anwendungen selbst erbracht werden. Dazu sind aber Eingriffe in die Anwendungen notwendig, was erstens entsprechende Zugriffsmöglichkeiten voraussetzt, die bei gekauften oder externen Anwendungen i.d.R. nicht vorliegen. Zweitens sind solche Eingriffe mit Aufwendungen verbunden und drittens führen sie zu einer höheren Abhängigkeit. Entsprechend angepasste Anwendungen können nicht ohne weiteres durch Alternativen ersetzt werden. Selbst die Einspielung eines neuen Releases des gleichen Produkts kann bereits mit erheblichen Problemen verbunden zu sein.

Analoge Überlegungen gelten für die nachfolgend vorgestellten Leistungen eines ESB.

Werden Verfahren abgebildet, bei denen verschiedene beteiligte Komponenten parallel arbeiten können?

1 siehe „Enterprise Service Bus Seite 81

2 siehe „nachrichtenorientierte Middleware“ auf Seite 82

Wie das Beispiel 10 ebenfalls zeigt, gibt es Verfahren, bei denen verschiedene Anwendungen gleichzeitig arbeiten. Die Persistierung durch das DMS und die Aufbereitung des Antrags für den Sachbearbeiter durch die Fachanwendung können parallel geschehen. Auch in diesem Fall bietet sich der Einsatz einer MOM an.

Wird Flexibilität hinsichtlich der Lokalisation von Services benötigt?

Eine MOM kann die Zuständigkeit für die Lokalisierung der Empfänger übernehmen. Die sendende Anwendung muss dann nur wissen, wie sie die MOM erreicht. Für die Lokalisierung der Empfänger ist dagegen die MOM zuständig. Ändert sich der Aufenthaltsort eines Services, muss nur die MOM umkonfiguriert werden, nicht aber sämtliche Anwendungen, die den Service nutzen.

Gibt es Ereignisse, bei deren Auftreten eine unbekannte Zahl von Services existiert, die über das Ereignis informiert werden müssen?

Eine MOM kann die Empfänger nicht nur lokalisieren, sondern auch mit Hilfe von Topics[1] entscheiden, an wen sie eine Nachricht schickt. Die sendende Anwendung muss nur entscheiden, zu welchem Topic die Nachricht gehört, die sie versendet. Wer als Interessent angemeldet ist und somit die Nachricht erhält, weiß sie nicht, und sie muss sich auch nicht darum kümmern.

Beispiel 11 Im Antragsverfahren für einen Gewerbeschein aus Beispiel 6 schickt die Fachanwendung des Landratsamts das Ergebnis der Prüfung an verschiedene Organisationen zur Kenntnisnahme weiter. Wird eine MOM eingesetzt, kann das Wissen darum, welche Organisationen zu informieren sind, aus der Fachanwendung des Landratsamts heraus gezogen werden. Dazu wird in der MOM ein entsprechendes Topic eingerichtet, an dem sich die Fachanwendungen der zu informierenden Organisationen registrieren. Die Fachanwendung des Landratsamts schickt dann eine Nachricht an dieses Topic, woraufhin die Fachanwendungen der anderen Organisationen automatisch die Nachricht erhalten.

[1] siehe „nachrichtenorientierte Middleware“ auf Seite 82

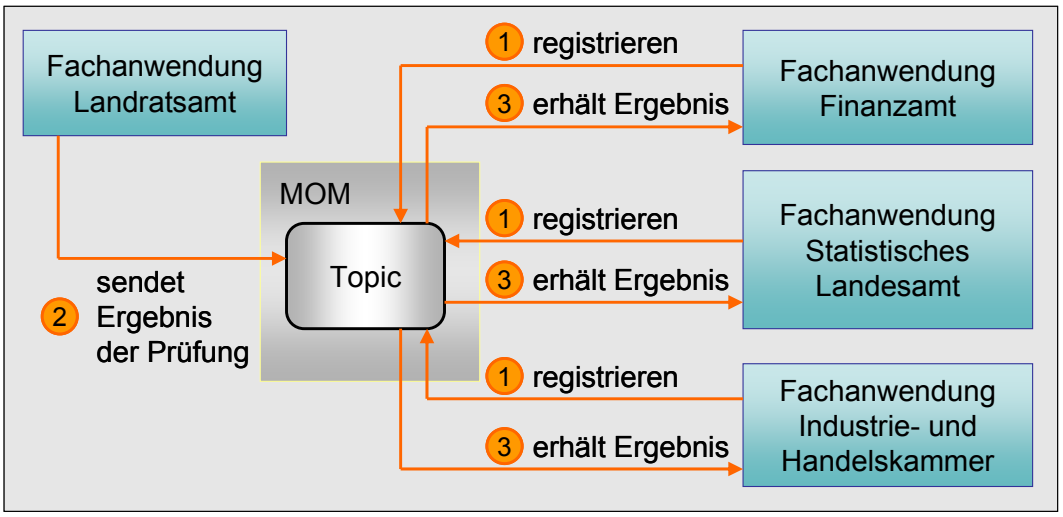


Abbildung 21: Einsatz eines Topics in einem Antragsverfahren

Kommen später weitere Interessenten hinzu, erhalten auch diese die Nachricht, ohne dass die sendende Anwendung angepasst werden muss. Müssen z.B. in Beispiel 11 aufgrund geänderter gesetzlicher Bestimmungen weitere Organisationen informiert werden, genügt es, wenn deren Fachanwendungen sich beim obigen Topic registrieren. Die Fachanwendung des Landratsamts muss in keiner Weise angepasst werden.

Müssen Legacy-Anwendungen (Altanwendungen) integriert werden?

Legacy-Anwendungen sind i.d.R. zu einem Zeitpunkt entwickelt worden, als an serviceorientierte Architekturen und organisationsübergreifende Integrationen noch gar nicht zu denken war. Sie sind daher nicht darauf zugeschnitten, mit anderen Anwendungen und Komponenten zu interagieren. Häufig decken sie aber wichtige Fachlogik ab, die nicht ohne weiteres durch andere Komponenten übernommen werden kann. Folglich müssen sie weiterhin genutzt und somit integriert werden.

Legacy-Anwendungen können entweder durch Anpassung der Anwendung selbst oder mit Hilfe eines ESB für eine serviceorientierte Architektur verfügbar gemacht werden. Im ersten Fall wird eine Service-Schnittstelle auf die Legacy-Anwendung aufgesetzt. Ob das ein gangbarer Weg ist, hängt von der konkreten Legacy-Anwendung ab und muss daher im Einzelfall entschieden werden. Da bei der Definition von Service-Schnittstellen mit Bedacht vorgegangen werden muss (siehe oben), kann dieses Vorgehen jedoch mit erheblichen Aufwendungen verbunden sein.

Im zweiten Fall wird der ESB mit Hilfe eines Adapters um die Möglichkeit erweitert, die Legacy-Anwendung zu integrieren. Ein Adapter versteckt die proprietäre Schnittstelle einer Legacy-Anwendung hinter einer Service-Schnittstelle. Dieses

Vorgehen ist auf jeden Fall dann vorzuziehen, wenn geplant ist, die Legacy-Anwendung in der näheren Zukunft durch eine andere Anwendung abzulösen, so dass sich Investitionen in die Legacy-Anwendung nicht mehr lohnen.

Wie die Ausführungen zeigen, gibt es zahlreiche Gründe, aus denen heraus sich der Einsatz eines ESB lohnen kann, auch wenn dieser i.d.R. mit nicht unerheblichen Beschaffungs- und Einführungskosten verbunden ist. Bei der Auswahl eines konkreten Produkts sind die allgemeinen Hinweise zur Auswahl von Integrationssoftware zu beachten (siehe Abschnitt 4.1.3.3).

Einen einheitlichen Standard für ESBs gibt es bisher nicht, wohl aber Standards bzw. Standardisierungsbemühungen, die die verschiedenen Probleme adressieren, die ein ESB löst:

- BPEL4WS (Business Process Execution Language for Web Services) und WS-CDL (Web Services Choreography Description Language) für die Orchestrierung von Web-Services. Von diesen hat BPEL4WS in SAGA den Status "unter Beobachtung" und WS-CDL steht in der White List[1].
- OSCI-Transport, XKMS (XML Key Management Specification), WS-Security und SAML (Security Assertion Markup Language) für den sicheren XML-basierten Dokumentenaustausch. Von diesen haben OSCI-Transport in SAGA den Status "obligatorisch" und WS-Security, SAML sowie XKMS den Status "empfohlen"[2].

Hinweise zum Einsatz eines Enterprise Service Buses:

- Wird mindestens eine der folgenden Fragen mit "Ja" beantwortet, sollte der Einsatz eines ESB[3] in Erwägung gezogen werden:
  - Werden änderungsanfällige Prozesse umgesetzt?
  - Müssen Services genutzt werden, die lange oder nicht vorhersagbare Antwortzeiten haben?
  - Sind Anwendungen oder Komponenten beteiligt, deren permanente Verfügbarkeit nicht garantiert ist?
  - Werden Verfahren abgebildet, bei denen verschiedene beteiligte Komponenten parallel arbeiten können?
  - Wird Flexibilität hinsichtlich der Lokalisation von Services benötigt?

1 Zur Bedeutung der Klassifizierung dieser Technologien siehe Kapitel „Klassifizierung und Lebenszyklen von Standards“ in SAGA 3.0 (www.kbst.bund.de)

2 Zur Bedeutung der Klassifizierung dieser Technologien siehe Kapitel „Klassifizierung und Lebenszyklen von Standards“ in SAGA 3.0 (www.kbst.bund.de)

3 siehe „Enterprise Service Bus Seite 81

- Gibt es Ereignisse, bei deren Auftreten eine unbekannte Zahl von Services existiert, die über das Ereignis informiert werden müssen?
- Müssen Legacy-Anwendungen (Altanwendungen) integriert werden?

- Wird ein ESB eingesetzt, sollte das ausgewählte Produkt den allgemeinen Anforderungen an eine Integrationssoftware genügen (siehe Abschnitt 4.1.3.3).
- BPEL4WS hat als Standard für die Orchestrierung von Web-Services in SAGA den Status "unter Beobachtung".
- OSCI-Transport hat als Standard für den sichern XML-basierten Dokumentenaustausch den Status "obligatorisch". WS-Security und XKMS haben den Status "empfohlen".

4.1.3.5 Integrationssoftware in komponentenbasierten Architekturen

Komponentenbasierte Architekturen (siehe Abschnitte 3.2.3 und 4.1.3.2) werden üblicherweise mit Hilfe einer Distributed Object Technology (DOT) umgesetzt. Beispiele für DOTs sind Enterprise Java Beans (EJB) aus der Java 2 Enterprise Edition (J2EE) und Microsofts DCOM/COM+. Kernelement einer DOT ist, dass Objekte einer Anwendung auf verschiedene Laufzeitumgebungen verteilt werden können. Die Verteilung erfolgt dabei jedoch in einer Art und Weise, die es den Objekten gestattet, Objekte aus einer anderen Laufzeitumgebung so anzusprechen, als befänden sie sich in der gleichen Laufzeitumgebung. Beispiele für Laufzeitumgebungen sind Virtual Machines im Falle von Java und Common Language Runtimes im Falle von .NET.

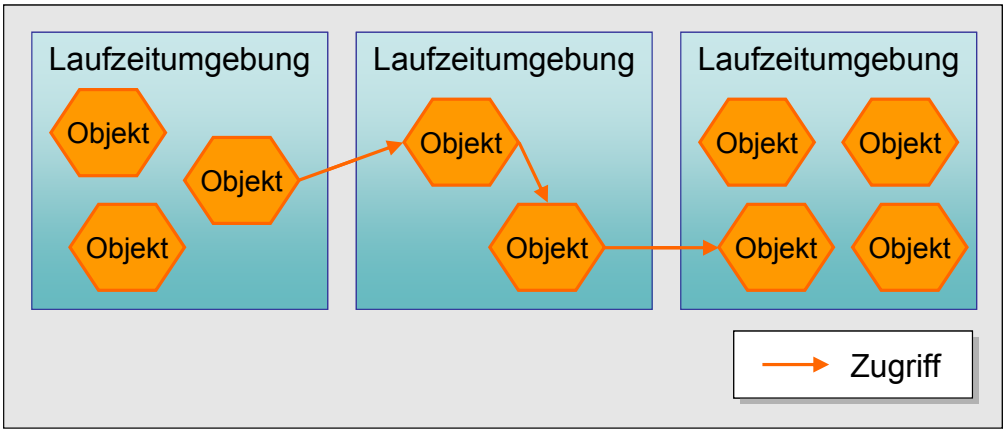


Abbildung 22: Verteilte Objekte

Die Umsetzung querschnittlicher Aspekte, wie z.B. Transaktions-, oder Persistenzmanagement, kann von spezieller Middleware, so genannter DOT-

Middleware übernommen werden. Beispiele für solche DOT-Middleware sind EJB-Server für die Java 2 Enterprise Edition (J2EE) und Common Language Runtimes für .NET. Der Sinn einer DOT-Middleware besteht darin, bestimmte, bei verteilten Objekten immer wieder benötigte Aufgaben zu übernehmen.

- *Organisation der Methodenaufrufe verteilter Objekte*
  Zu diesem Zweck lokalisiert die Middleware das Empfängerobjekt (meist in einer anderen Laufzeitumgebung), überträgt den Methodenaufruf an das Empfängerobjekt und liefert gegebenenfalls die Antwort an das aufrufende Objekt zurück.
- *Organisation des Lifecycles von verteilten Objekten*
  Die Middleware erzeugt, aktiviert, deaktiviert und zerstört verteilte Objekte.
- *Organisation konkurrierender Zugriffe*
  Wird ein Objekt von mehreren Objekten gleichzeitig angesprochen, kann dies zu einer unerwünschten gegenseitigen Beeinflussung dieser Zugriffe führen. DOT-Middleware übernimmt die Aufgabe, diese konkurrierenden Zugriffe so zu koordinieren, dass keine gegenseitige Beeinflussung entsteht, beispielsweise durch Sequenzialisierung der Aufrufe oder durch Bereitstellung mehrerer gleichartiger Objekte.
- *Transaktionsverwaltung*
  DOT-Middleware kann sich darum kümmern, dass inhaltlich zusammengehörende Operationen in so genannten Transaktionen gebündelt werden.
- *Persistenzmanagement*
  DOT-Middleware kann sich um die Persistierung von Daten kümmern. Die Art und Weise, wie die Daten persistiert werden (z.B. in einer relationalen Datenbank, in einem LDAP-Verzeichnis oder als Dateien in einem Dateiverzeichnis) ist dann für die Komponenten transparent, kann ohne Anpassung der Komponenten durch eine andere Alternative ersetzt werden.

Der Einsatz einer DOT-Middleware ist zwar mit einmaligen Kosten für die Anschaffung verbunden. Diese Investition lohnt sich jedoch meistens, da die von der DOT-Middleware bereit gestellten Dienste fast immer benötigt werden und somit anderenfalls selbst implementiert werden müssten. Darüber hinaus vermindert der Einsatz einer DOT-Middleware die Plattformabhängigkeit der Anwendungen. Wird beispielsweise das Persistenzmanagement von einer DOT-Middleware übernommen, kann der verwendete Persistenzmechanismus leichter ausgetauscht werden.

Es ergibt sich natürlich eine gewisse Abhängigkeit vom konkret ausgewählten DOT-Middleware-Produkt. Die sich aus dieser Abhängigkeit ergebenden Konsequenzen gelten aber für alle Arten von Integrationssoftware und wurden daher bereits in Abschnitt 4.1.3.3 behandelt.

Hinweise zum Einsatz von DOT-Middleware

- Die verwendete DOT-Middleware sollte den allgemeinen Anforderungen an Integrationssoftware genügen (siehe Abschnitt 4.1.3.3).
- Bei der Beschaffung oder Entwicklung zukünftiger Komponenten für die Umsetzung des Services ist darauf zu achten, dass diese mit Hilfe derjenigen Technologie entwickelt werden, die der DOT-Middleware zugrunde liegt, und nicht auf ein bestimmtes DOT-Middleware-Produkt angewiesen sind, weil sie beispielsweise proprietäre Funktionalitäten dieses Produkts verwenden.

4.1.3.6 Fazit zu Inhalten einer Gesamtarchitektur

Wie die vorangehenden Ausführungen gezeigt haben, ist der Aufbau einer Systemarchitektur, bei der einerseits die einzelnen Anwendungen so integriert sind, dass die medienbruchfreie Umsetzung der behördlichen Prozesse möglich ist, und andererseits Abhängigkeiten vermieden werden, die die Weiterentwicklung der IT-Landschaft erschweren würden, keine triviale Aufgabe. Verschiedene Architekturtypen stehen zur Auswahl, die jeweils ihre Stärken und Schwächen haben. In diesem Abschnitt werden die wichtigsten Entscheidungshilfen für die Auswahl von Architekturtypen und Infrastruktursoftware-Typen zusammen gefasst.

Silo-Architekturen, bei denen die einzelnen Fachanwendungen als Monolithen realisiert werden, kommen nur dann in Betracht, wenn es keinerlei fachliche Abhängigkeiten zwischen den verschiedenen Anwendungen gibt, oder wenn nichttechnische Gründe, z.B. datenschutzrechtliche Gründe, gegen eine Integration der Anwendungen sprechen (siehe Kapitel 4.1.3.1).

Eine serviceorientierte Architektur (SOA), bei der die Fachlogik mit Hilfe einer Reihe von lose gekoppelten, unabhängigen Services umgesetzt wird, bietet sich dann an, wenn jetzt oder in absehbarer Zukunft

- Anwendungen integriert werden müssen, die mit Hilfe unterschiedlicher Technologiefamilien realisiert wurden (siehe Kapitel 4.1.3.2).
- Anwendungen integriert werden müssen, die von unterschiedlichen Behörden oder unterschiedlichen, jeweils autarken Abteilungen innerhalb einer Behörde betrieben werden (siehe Kapitel 4.1.3.2).

Die übliche Technologiefamilie für die Umsetzung einer SOA ist die Web-Service-Technologiefamilie mit ihren Bestandteilen SOAP, WSDL, UDDI und XSD (siehe Kapitel 4.1.3.2, Seite 73).

Bei der Definition der Schnittstellen einer serviceorientierten Architektur ist zu beachten, dass Services grobgranulare Schnittstellen und kontextfreie Funktionalitäten anbieten sollten (siehe Kapitel 4.1.3.2, Seite 73).

Entwicklung und Beschaffung (plattform)unabhängiger Fachanwendungen

Für die Realisierung der einzelnen Services einer SOA bieten sich komponentenbasierte Architekturen an, bei denen die Fachlogik mit Hilfe eng gekoppelter, teilweise zustandsbehafteter Komponenten umgesetzt wird (siehe Kapitel 4.1.3.2, Seite 75). Dasselbe gilt für die Realisierung vollständiger Fachanwendungen, die entweder gar nicht mit anderen Anwendungen und Komponenten interagieren oder nur mit solchen, die mit Hilfe der gleichen Technologiefamilie implementiert wurden und deren weitere Entwicklung entsprechend beeinflusst werden kann.

Services, für deren Realisierung Komponenten benötigt werden, bei denen kein Einfluss auf die Implementierungstechnologie genommen werden kann, sollten in Teilservices zerlegt werden, die dann im Wege einer serviceorientierten Architektur integriert werden (siehe Kapitel 4.1.3.2, Seite 76).

Die verschiedenen Architekturtypen können kombiniert werden. Auch wenn die Gesamtarchitektur serviceorientiert ist, können einzelne Anwendungen als Monolithen realisiert werden, sofern sie keine fachlichen Abhängigkeiten zu anderen Anwendungen aufweisen. Serviceorientierte und komponentenbasierte Architekturen spielen oft zusammen, wobei die serviceorientierte Architektur die Integration "im Großen" regelt und die Implementierung der einzelnen Services komponentenbasiert erfolgt.

Müssen viele Anwendungen integriert werden, bietet sich der Einsatz einer Integrationssoftware an (siehe Kapitel 4.1.3.3 Seite 77). Eine solche Integrationssoftware steuert die einzelnen Bausteine und verhindert so das Entstehen einer Stovepipe-Architektur. Für die Integration von Services können Enterprise Service Buses verwendet werden, während die Integration von Komponenten mit Hilfe von DOT- Middleware erfolgt. Ebenso, wie man serviceorientierte Architekturen und komponentenbasierte Architekturen kombinieren kann, können Enterprise Service Buses und DOT-Middleware gemeinsam eingesetzt werden.

Die nachfolgende Abbildung 22 fasst noch einmal die verschiedenen Architektur- und Integrationssoftware-Typen in einer Grafik zusammen.

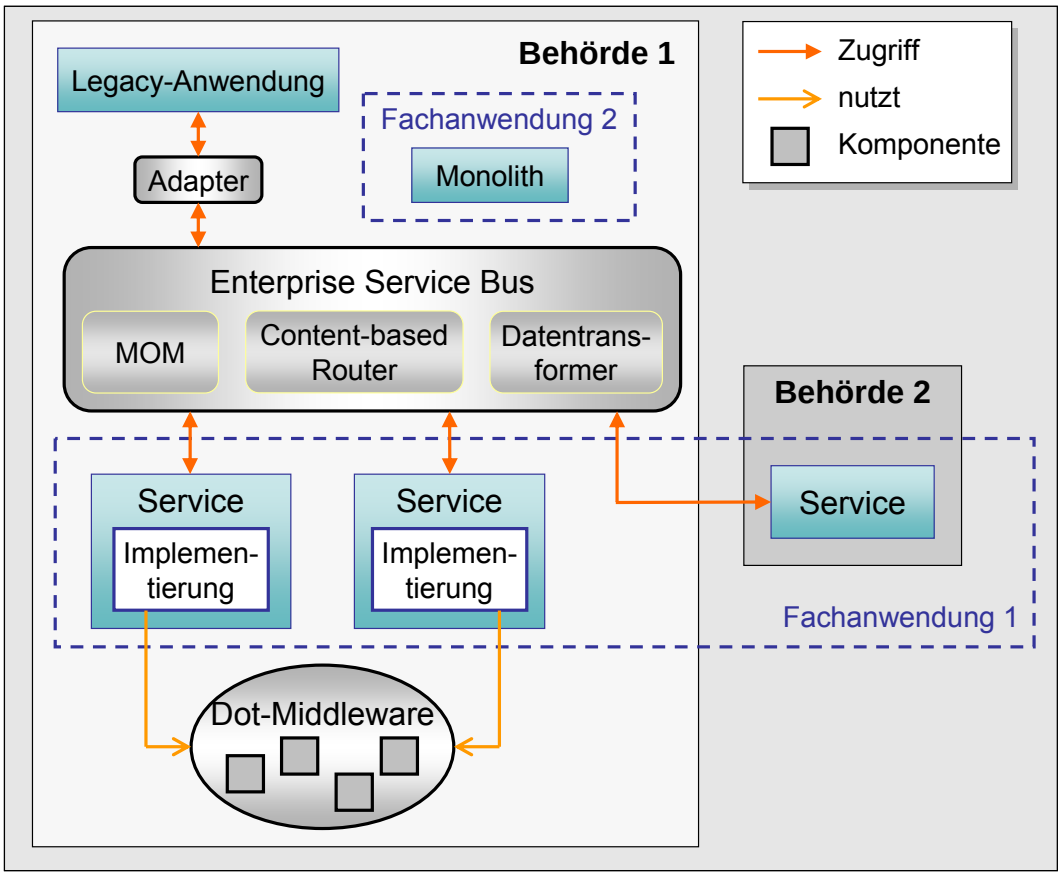


Abbildung 23: Architektur- und Integrationssoftware-Typen

4.1.4 Personal und Know-how

Einschlägiges Know-how kann bei der Beschaffung einer Fachanwendung unterschiedliche Rollen spielen, abhängig davon, ob die Anwendung 'off the shelf' gekauft oder intern bzw. extern entwickelt wird. Jede Behörde muss somit die Fragen beantworten, wie benötigtes, jedoch noch nicht vorhandenes, Know-how aufgebaut werden kann, und wie bereits vorhandenes Know-how in eine Richtung gelenkt werden kann, die im Einklang mit der IT-Strategie steht.

4.1.4.1 Einführung neuer Technologien

Ein spezielles Problem, das sowohl beim Kauf als auch bei der Entwicklung einer Fachanwendung auftreten kann, ist die Einführung einer neuen Technologie. Soll eine neue Fachanwendung mit besonderem Augenmerk auf ihre Plattformunabhängigkeit und ihre Korrespondenz mit der Integrationsarchitektur gekauft oder entwickelt werden, dann erfordert sie möglicherweise den Einsatz anderer Technologien und -Architekturen als die bisher verwendeten.

Entwicklung und Beschaffung (plattform)unabhängiger Fachanwendungen

Die Einführung neuer Technologien ist häufig ein recht zähes Unterfangen. "Slow Technology Transfer" begünstigt nach Capers Jones[1] die Materialisierung einer ganzen Reihe von Projektrisiken, in erster Linie terminlicher Art. Seiner Beobachtung nach verläuft die Einführung neuer Technologien in der Privatwirtschaft schneller als in der Öffentlichen Verwaltung und beim Militär, sie gelingt in zentralisierten Organisationen zügiger als in dezentralen, und sie fällt dem technischen Personal leichter als dem Software-Management[2].

Der von Jones als Richtwert angegebenen Zeitbedarf für den "Technology Transfer" hängt von der Größe der Zielgruppe ab. Besteht die Gruppe aus einem einzigen Mitglied, beträgt der Zeitbedarf einen bis sechs Monate. Umfasst die Gruppe 10 Mitglieder, beträgt er 6 bis 12 Monate. Eine Gruppe mit 100 Mitgliedern benötigt 12 bis 24 Monate. Eine große Gruppe ist also schneller als viele kleine Gruppen nacheinander. Gemessen an der Dauer üblicher Software-Entwicklungprojekte sind diese Zeiträume durchaus beachtlich und verdeutlichen das Risiko des Technologiewechsels in einem kritischen Projekt.

Um die Risiken zu minimieren, sollte die Einführung einer neuen Technologie als selbständige Aufgabe begriffen werden und die Technologie zunächst an einem unkritischen Gegenstand ausprobiert werden. Dabei kann aus Jones' Zahlen ein zweistufiges Vorgehen abgeleitet werden. In der ersten Stufe eignet sich eine kleine Gruppe die Technologie an. In der zweiten Stufe wird die Technologie breit eingeführt, wobei der genannten Gruppe die Rolle der "reference facility" zufällt. Ferner kann die Einführung neuer Technologien dadurch beschleunigt werden, dass die Organisation Bibliotheken und zentrale Anlaufpunkte ("reference facilities") etabliert sowie Trainings durchführt und dafür geeignete Trainingsobjekte anbietet.

Eine schlüssige Software-Strategie, deren Bestandteil das "strategische Abhängigkeitsmanagement" ist, kann bei der Plausibilisierung der Kosten helfen, die mit der expliziten Adressierung der neuen Technologie verbunden sind.

Um beispielsweise J2EE tatsächlich effektiv zu nutzen, ist neben der Beherrschung der Technologie im engeren Sinne darüber hinaus auch das Verständnis des assoziierten Rollen- und Vorgehensmodells sowie zahlreicher Anwendungsmuster je nach Projektkontext hilfreich, wenn nicht unverzichtbar. Wie die Praxis gezeigt hat, besteht bei Missachtung dieser Anwendungsmuster ein erhebliches Risiko, unakzeptabel träge Systeme zu erzeugen.

Bei den ersten Einsätzen neuer Technologien in kritischen Projekten empfiehlt sich daher ein dediziertes Risiko-Management mit Blick auf die Konsequenzen

---

[1] Jones, Capers: Assessment and Control of Software Risks. Upper Saddle River, NJ (Prentice Hall PTR) 1994, Seite 534 ff.

[2] Software-Management meint in diesem Kontext das Personal, welches für das Management des gesamten Software-Lebenszyklus verantwortlich ist (betrifft z.B. Entscheidungen über Weiterentwicklungsstrategien u.a.).

des Technologieeinsatzes. Insbesondere ist die frühzeitige Entwicklung von vertikal vollständig integrierten Prototypen ("Durchstich") ratsam, ihre Befüllung mit realistischerweise erwartbaren Datenmengen und die Prüfung ihrer Antwortzeiten. Eine solche Verifikation der Architektur sollte auch und gerade dann stattfinden, wenn das 'gefühlte Verständnis' für die neue Technologie bereits besonders ausgeprägt ist.

4.1.4.2 Einbindung skriptender Power-User

Einige Organisationen betreiben Software-Entwicklung, ohne sich darüber voll bewusst zu sein. Solche Organisationen haben keine Strukturen und kein ausgewiesenes Personal für diese Aufgabenstellung. Findige 'Power User' verwenden jedoch die Möglichkeiten der verfügbaren Anwendungen, um Programme zu schreiben. Die dazu verwendeten Werkzeuge sind in erster Linie die Makro- und Scripting-Mechanismen von Office-Suiten. Die entwickelten Programme sind i.d.R. zunächst sehr überschaubar und auf den persönlichen Bedarf zugeschnitten, finden jedoch mitunter auch größere Verbreitung und wachsen in der Konsequenz stetig an.

Unter dem Gesichtspunkt der Plattformunabhängigkeit sind solche Programme i.d.R. ausgesprochen problematisch. Diese Probleme sind teilweise offensichtlich. Ein Programm, das in der Skriptsprache einer Office-Anwendung geschrieben wurde, läuft möglicherweise bereits auf der nächsten Version der gleichen Office-Anwendung nicht mehr wie intendiert, weil sich die Sprache oder das Office-Dokumentenmodell geändert hat, und es läuft natürlich auch auf keiner anderen Plattform. Andere Probleme offenbaren sich erst auf den zweiten Blick. Viele Office-Skripte sind eng mit Merkmalen der Ausführungsumgebung verzahnt oder hängen von Dokumentenvorlagen ab, ohne jedoch in diese eingebettet zu sein. Ändert sich die Umgebung oder die Vorlage, versagt das Programm seinen Dienst. Die Pflege oder notfalls die Portierung solcher Programme wird in aller Regel erheblich dadurch erschwert, dass ihr intendiertes Verhalten nicht spezifiziert und ihre Abhängigkeiten nicht dokumentiert sind. Office-Makros erweisen sich so immer wieder als ein teilweise unüberwindbares Migrationshindernis.

Allerdings ist die starke Verbreitung solcher Programme, die sie erst problematisch macht, zugleich ein Indiz dafür, dass sie auf die eine oder andere Weise zur technischen Unterstützung von Prozessen beitragen. Zudem verfügen die Personen, die solche Programme entwickeln, augenscheinlich über zumindest rudimentäre Programmierkenntnisse.

Völlig unterbinden lässt sich derartige „Graswurzel-Softwareentwicklung“ kaum. Im Sinne der Förderung von Plattformunabhängigkeit stellen sich daher zwei Fragen:

- Wenn diese Art der Softwareentwicklung schon stattfindet, wie können dann die damit einher gehenden Plattformbindungen reduziert werden?
- Wie lässt sich diese Art der Softwareentwicklung möglicherweise im Rahmen der Beschaffung von plattformunabhängigen Fachanwendungen nutzen?

Plattformunabhängigkeit ist wie bereits angesprochen, eine strategische Gestaltungsaufgabe, Skript-Programmierung ist dagegen taktisch motiviert. Um mit einer Strategie in Einklang gebracht werden zu können, muss sie strukturell adressierbar gemacht werden. Ein erster Schritt dazu kann sein, die Skript-Programmierer zu einer Gruppe zusammenzuführen. Eine solche Gruppe kann rein informell sein und sie kann Personen aus unterschiedlichen organisatorischen Einheiten bündeln.

Diese Bündelung ist in erster Linie notwendig, um die bis dahin ungesteuerten Entwicklungsaktivitäten effektiv steuerbar zu machen und sie auf diese Weise mit der IT-Strategie und ihren Zielen in Einklang zu bringen, namentlich Plattformunabhängigkeit.

Die Steuerung und Bündelung von Aktivitäten im Bereich der Skript- und Makro-Programmierung soll durch die Entwicklung von verbindlichen Regelungen und Handlungsanweisungen untermauert werden. Zu den wesentlichen Regelungen und Handlungsanweisungen gehören in diesem Kontext:

- *Wartbarkeit*
  Softwareentwicklung ist mehr als Programmierung und unterliegt einem Lebenszyklus, in dessen Verlauf sie korrigiert, angepasst und weiterentwickelt sowie möglicherweise auf andere Plattformen verlegt wird. Es ist im Vorfeld meist nicht bekannt, wann eine dieser Maßnahmen notwendig wird und wer sie ausführt. Die Software sollte daher so gestaltet werden, dass sie auf diesen Lebenszyklus vorbereitet ist. Beispielsweise müssen die Quelltexte verständlich geschrieben sein, nicht nur für den ursprünglichen Programmierer, sondern auch für denjenigen, der dessen Arbeit zu einem späteren Zeitpunkt fortführt. Über die Verständlichkeit auf der Ebene des Quellcodes hinaus wird ebenfalls die auf der der Ebene des Entwurfes verlangt.
- *Kohärenz und Abhängigkeiten*
  Zwei weitere Grundsätze sind die Trennung von Schnittstellen und Implementierungen sowie der Bündelung von Abhängigkeiten. Beide liefern Strukturen, anhand derer Programmcodes in jeweils relevante und vernachlässigbare Teile unterteilt werden können, so dass die jeweils in Be-

tracht zu ziehenden Programmteile überschaubar bleiben und unabhängig voneinander bearbeitet werden können.

- *Abstraktion*
  Die Forderung, Quelltexte mögen verständlich geschrieben sein, soll nicht heißen, dass Quelltexte der bevorzugte oder gar einzige Kommunikationskanal sein sollen. Wie in den Ausführungen über die Einflussnahmemöglichkeiten auf den Entwicklungsprozess (siehe Abschnitt 4.2.1.2) dargelegt, produzieren Vorgehensmodelle der Softwareentwicklung in erster Linie Modelle als fokussierte Abstraktionen des zu entwickelnden Systems.

- *Kommunikation und Notation*
  Abstraktionen sind für einen individuellen Softwareentwickler hilfreich, um sich über seine Vorstellungen vom zu entwickelnden System klar zu werden. Ihre hauptsächlich beabsichtigte Verwendung ist aber die Kommunikation. Als Standard-Notation für Modelle in der Softwareentwicklung und in vielen angrenzenden Themenfeldern hat sich die Unified Modelling Language (UML) etabliert. Ihre Verwendung erleichtert die Kommunikation von Abstraktionen ganz erheblich und macht sie in informellen Strukturen oft erst sinnvoll möglich.

Diese Ziele werden durch Werkzeuge und Methoden unterstützt. Der Quellcode soll – wenn möglich – an zentraler Stelle versioniert werden. Jedes Stück Quellcode soll nach Möglichkeit einmal einer gemeinsamen Begutachtungen unterzogen werden. Solche Begutachtungen sind eines der effektivsten Mittel zur Beseitigung von Fehlern in Software. Sie können methodisch unterschiedlich untermauert werden ("Code Review", "Peer Review", "Walkthrough"). Ihr Nutzen entfaltet sich jedoch auch ganz formlos. Er beruht auf den Prinzip, dass vier Augen mehr sehen als zwei. Besonders für ein Team von Skript-Entwicklern haben Begutachtungen zahlreiche weitere Vorteile. Erstes Ziel einer Begutachtung ist das Verständnis. So geht von der Begutachtung ein gewisser Anreiz aus, verständlich zu programmieren, sinnvoll zu kommentieren und eventuelle Konventionen zu befolgen. Zugleich fördert eine Begutachtung die Kommunikation über den Quellcode und über den Prozess des Programmierens und führt dadurch zu einem besseren Prozessverständnis. Schließlich sind Begutachtungen eine ideale Möglichkeit, um Ideen auszutauschen und voneinander zu lernen.

Neben der Möglichkeit, das Engagement interessierter Mitarbeiter durch Reviews zu ermutigen, hat eine Organisation selbstverständlich stets die Möglichkeit, ihnen kommerzielle Seminare angedeihen zu lassen und sie durch die Zertifizierungsverfahren von Produkt-Anbieter, Projektmanagement-Instituten und Prozess-Auditoren zu schleusen.

4.2 Von der Anforderung zur Anwendung

Kapitel 3 hat aufgezeigt, wie Anwendungsarchitekturen optimiert werden können. Die vorangehenden Abschnitte dieses Kapitels sind der Frage nachgegangen, wie in der Vielfalt des Wünschenswerten das Machbare und das vor dem Hintergrund einer IT-Strategie Richtige identifiziert werden kann. In diesem Abschnitt werden Hinweise gegeben, wie im Rahmen der Beschaffung oder der Eigenentwicklung einer Fachanwendung dafür gesorgt werden kann, dass diese auch diesen Anforderungen genügt. Wie bereits zu Beginn des Kapitels 4 deutlich gemacht wurde, kann es sich bei einer Beschaffung um den Kauf eines „fertigen" Standardproduktes[1] oder um die Beauftragung der Entwicklung einer Individualanwendung handeln. Welche dieser drei Varianten zu wählen ist, hängt erstens davon ab, ob überhaupt passende Standardprodukte zur Verfügung stehen, zweitens, ob für eine Eigenentwicklung das entsprechende Personal mit ausreichendem Know-how zur Verfügung steht und drittens, in welchem der drei Fälle auf die wirtschaftlichste Weise beschafft werden kann. Hierzu müssen entsprechende Maßnahmen im Vorfeld der Beschaffung durchgeführt werden. Zum einen ist eine Marktanalyse hinsichtlich der Verfügbarkeit von Standardprodukten durchzuführen. Um diese durchführen zu können, müssen natürlich zunächst die Anforderungen an die Fachanwendung definiert werden. Sollte sowohl der Kauf eines Standardproduktes, wegen fehlender Verfügbarkeit eines passenden Produktes, als auch die Eigenentwicklung, wegen fehlender Verfügbarkeit des passenden Personals nicht in Frage kommen, muss trotzdem eine Wirtschaftlichkeitsbetrachtung im Sinne der Betrachtung von Alternativen erfolgen. Bezogen auf den Fokus dieses Leitfadens wäre dies die Betrachtung der Alternativen plattformunabhängig und plattformabhängig, was aber nicht die einzigen sein müssen.

Kommen nur die Varianten Kauf eines „fertigen" Standardproduktes oder Auftragsentwicklung in Frage, so muss eine Kostenabschätzung mittels Wirtschaftlichkeitsbetrachtung vorgenommen werden, um festlegen zu können, welche Handlungsoption gewählt wird.

Im Folgenden soll dem Leser keine erneute allgemeine Erläuterung des Beschaffungsprozesses mit Wirtschaftlichkeitsbetrachtung, Ausschreibung usw. präsentiert werden, denn hierzu stellt die Koordinierungs- und Beratungsstelle der Bundesregierung für Informationstechnik in der Bundesverwaltung (KBSt) bereits verschiedene Dokumente und Werkzeuge auf ihren Web-Seiten zur Verfügung. Unter anderem sind dies:

[1] Bei den wenigsten Fachanwendungen, die als Standardprodukt angeboten werden, kann man von fertigen Produkten reden. Fast alle derartigen Fachanwendungen bedürfen einer Anpassung an die speziellen fachlichen Anforderungen einer Behörde. Dies hat aber nichts mit einer Anpassung hinsichtlich des Grades der Plattformunabhängigkeit zu tun. Diese Eigenschaft, in welcher Form auch immer, ist schon durch die Architektur der Fachanwendung festgelegt worden.

- die „Unterlage für die Ausschreibung und Bewertung von IT-Leistungen“ (UfAB III) in der jeweils aktuellen Fassung[1], Version 2.0[2], diese liefert u.a. Hinweise, wann Wirtschaftlichkeitsbetrachtungen durchzuführen und wann Ausschreibungen und in welcher Form durchzuführen sind. Weiterhin beschäftigt sich die UfAB methodisch mit der Bewertung von Angeboten.
- die WiBe 4.0 mit ihren „Empfehlung zur Durchführung von Wirtschaftlichkeitsbetrachtungen in der Bundesverwaltung, insbesondere beim Einsatz der IT“.

Die nächsten Abschnitte beschäftigen sich damit, wie die Anforderungen zur Plattformunabhängigkeit im Rahmen dieses Beschaffungsprozesses Berücksichtigung finden können bzw. müssen.

Dabei wird auf die technischen, wirtschaftlichen und rechtlichen Aspekte des Beschaffungsprozesses eingegangen. Die technischen Aspekte betrachten vor allem die technische Spezifikation der Plattformunabhängigkeit im Rahmen der Anforderungsdefinition bzw. Leistungsbeschreibung sowie die Berücksichtigung der Kriterien zur Plattformunabhängigkeit im Rahmen der Bewertung von Angeboten nach UfAB III. Die wirtschaftlichen Aspekte behandeln die Thematik der Plattformunabhängigkeit aus Sicht der Wirtschaftlichkeitsbetrachtung nach WiBe 4.0. Die rechtlichen Aspekte spielen nur dann eine Rolle, wenn es darum geht, die Beschaffung im Rahmen einer Ausschreibung durchzuführen. Die rechtlichen Aspekte zeigen dabei auf, was aus rechtlicher Sicht, insbesondere mit Blick auf die Definition von Anforderungen, zu berücksichtigen ist.

4.2.1 Technische Aspekte

Im ersten Teilabschnitt der technischen Aspekte werden Hinweise gegeben, die sowohl den Kauf als auch die Entwicklung (Eigenentwicklung oder Auftragsentwicklung) einer Fachanwendung betreffen. In den nachfolgenden Teilabschnitten werden dann die Besonderheiten von Kauf und Entwicklung behandelt.

4.2.1.1 Spezifikationstreue

Der zentrale Begriff hinter der Frage, wie sichergestellt werden kann, dass eine Anwendung ihren Anforderungen auch genügt, ist Spezifikationstreue.

Die Vorbedingung für Spezifikationstreue ist, dass es eine Spezifikation gibt. Nur für diejenigen Anforderungen, die Bestandteil der Spezifikation sind, kann erwartet werden, dass sie am Ende auch erfüllt werden. Daraus ergibt sich die aus

[1] siehe http://www.kbst.bund.de/ „Wirtschaftlichkeit und Recht / UfAB“

[2] siehe http://www.kbst.bund.de/ „Wirtschaftlichkeit und Recht / UfAB“

dem Blickwinkel der Plattformunabhängigkeit wichtigste Regel für den Kauf oder die Entwicklung von Fachanwendungen:

> *Anforderungen, die dem Ziele der Reduzierung von Plattformabhängigkeit dienen, gehören in die Spezifikation.*

Diese Anforderungen treten damit in eine Reihe mit weiteren nichtfunktionalen Anforderungen, wie z.B. Anforderungen hinsichtlich Performanz, Sicherheit, Skalierbarkeit, Konfigurierbarkeit, Monitoring und Wartbarkeit. Auch aus rechtlicher Sicht ist die Aufnahme dieser Anforderungen nicht nur zulässig (siehe Abschnitt 4.2.3.4), sondern sogar notwendig, denn Kriterien, die in der Ausschreibung nicht genannt werden, dürfen später bei der Entscheidung für ein konkretes Produkt oder – im Falle der Auftragsentwicklung – einen konkreten Anbieter nicht berücksichtigt werden (siehe Abschnitt 4.2.3.3).

Die Formulierung von Anforderungen ist jedoch bekanntlich nicht immer ganz einfach, nicht ohne Grund wurden schon zahlreiche Bücher darüber geschrieben. Anforderungen, die die Plattformunabhängigkeit betreffen, sind da keine Ausnahme. An dieser Stelle sei auf zwei besonders wichtige Punkte hingewiesen:

- Jede Anforderung verfolgt ein Ziel. Es ist dieses Ziel, auf das es ankommt, nicht aber der Weg, wie das Ziel erreicht wird.
- Je präziser die Anforderungsbeschreibung, desto leichter lässt sich die Anforderung am fertigen Produkt überprüfen. Zusammen mit dem vorhergehenden Punkt folgt, dass das Ziel der Anforderung so präzise beschrieben werden sollte, dass leicht überprüft werden kann, ob es erreicht wurde.

Eine gute Spezifikation benötigt dreierlei Dinge:

- eine sehr präzise Vorstellung von der Beschaffenheit der fertigen Fachanwendung,
- erfahrene technische Autoren sowie
- eine sorgfältige und selbstkritische Qualitätskontrolle.

Es spielt dabei keine Rolle, aus welcher Nomenklatur der Name einer Spezifikation entlehnt wird und ob sie nun "Fachfeinkonzept" oder "Implementationsmodell" genannt wird.

Aus dem Blickwinkel dieses Leitfadens ist das Ziel die Plattformunabhängigkeit. Es reicht allerdings nicht aus, die Anforderung "Die Anwendung soll plattformunabhängig sein" in die Spezifikation zu schreiben. Diese Anforderung beschreibt zwar das Ziel und erfüllt insofern den ersten Punkt. Sie ist jedoch bei weitem nicht präzise genug, da sie nicht erklärt, was unter Plattformunabhängigkeit verstanden wird. In den vorangehenden Abschnitten und Kapiteln wurde aufgezeigt, welche unterschiedlichen Ausprägungen von Plattformabhängigkeit es gibt und wo Plattformunabhängigkeit in Widerstreit mit anderen Anforderungen treten

kann. Basierend auf diesem Wissen muss zunächst festgelegt werden, bezüglich welcher Plattformen und in jeweils welchem Maße Unabhängigkeit gewünscht wird. Diese Festlegungen müssen dann als entsprechende Anforderungen formuliert werden. Dazu einige Beispiele:

Beispiel 12 Die Behörde X möchte eine neue Fachanwendung anschaffen. Einige der Mitarbeiter der Behörde verwenden Debian Linux-Clients und andere Windows 2000/XP Clients.

Das Ziel der Unabhängigkeit fokussiert sich bezüglich der Plattform in diesem Fall auf das Betriebssystem. Die neue Anwendung muss mindestens auf den Betriebssystemen Debian Linux, Windows 2000 und Windows XP lauffähig sein. Eine Anforderung, die dieses Ziel ausdrückt, könnte beispielsweise wie folgt aussehen:

> *Die Anwendung muss auf verschiedenen Betriebssystemen lauffähig sein. Mindestens müssen die Betriebssysteme Debian Linux, Windows 2000 und Windows XP unterstützt werden.*

Eine Technologieplattform, die dieses Ziel unterstützt, ist die Java-Plattform. Java-Anwendungen sind auf verschiedenen Betriebssystemen lauffähig, insbesondere den oben aufgeführten. Der Grund dafür ist, dass in Java codierte Anwendungen vom Compiler in betriebssystemunabhängigen Java-Bytecode übersetzt werden. Dieser Bytecode wird dann von einer auf dem jeweiligen Betriebssystem laufenden Java Virtual Machine (JVM) interpretiert und ausgeführt. JVMs gibt es für eine Vielzahl von Betriebssystemen und es darf davon ausgegangen werden, dass es sie auch für zukünftige Betriebssysteme geben wird, sofern diese ein Mindestmaß an Verbreitung finden. Somit ist sichergestellt, dass auch die Portierung auf zukünftige Betriebssysteme ohne Anpassung der Anwendung möglich sein wird.

Die Behörde könnte daher auf die Idee kommen, die Verwendung von Java als Programmiersprache als Anforderung zu formulieren. Das wäre jedoch aus folgenden Gründen nicht sinnvoll:

- Erstens wäre es eine unnötig starke Einschränkung. Wie oben dargelegt, ergibt sich die Plattformunabhängigkeit einer Java-Anwendung aus dem Bytecode, nicht aber aus dem Quellcode. Java-Bytecode kann grundsätzlich auch aus einem Quellcode erzeugt werden, der in einer anderen Programmiersprache geschrieben wurde. Ein Anbieter, dessen Mitarbeiter mit einer anderen Programmiersprache arbeiten, der aber Java-Bytecode mittels eines entsprechenden Compilers erzeugt, liefert Anwendungen, die gleichermaßen plattformunabhängig sind. Es gibt keinen Grund, weshalb ein solcher Anbieter von vornherein ausgeschlossen werden sollte.

Darüber hinaus bieten auch andere Programmiersprachen Unabhängigkeit bezüglich des Betriebssystems (siehe auch Abschnitt 3.1.4). Dabei sind die Empfehlungen aus SAGA hinsichtlich der zu verwendenden Programmiersprachen zu beachten[1].

- Zweitens garantiert die Verwendung einer Programmiersprache, die Plattformunabhängigkeit grundsätzlich unterstützt, noch nicht, dass die resultierende Anwendung tatsächlich plattformunabhängig ist, wie in Kapitel 3.1.4 dargelegt wurde.
- Drittens sprechen rechtliche Gründe dagegen. Nach dem Gleichbehandlungsgrundsatz dürfen technische Spezifizierungen nämlich nur dann in Ausschreibungen gefordert werden, wenn hierfür ein sachlicher Grund besteht (siehe 4.2.3.2). Ein solcher sachlicher Grund liegt hier aber, wie oben dargelegt, nicht vor.[2]

Die beschriebenen Probleme können hier deshalb auftreten, weil gegen die erste der oben aufgeführten Regeln verstoßen wurde. Es wurde nicht das Ziel einer Anforderung spezifiziert, sondern ein möglicher Weg, wie dieses Ziel erreicht werden kann. Das Ziel ist hier die Unabhängigkeit vom Betriebssystem. Die Verwendung von Java ist dagegen nur ein möglicher Weg, dieses Ziel zu erreichen.

Beispiel 13 Die Behörde Y möchte ebenfalls eine neue Fachanwendung anschaffen. Auch die Mitarbeiter von Y verwenden unterschiedliche Betriebssysteme, und die neue Anwendung soll für länger Zeit eingesetzt werden. Anders als die Behörde X verfügt die Behörde Y über ein eigenes Team von Java-Entwicklern und plant, die Anwendung eigenständig weiterzuentwickeln.

Auch hier ist die Plattformunabhängigkeit bezüglich des Betriebssystems ein Ziel. Darüber hinaus soll die Anwendung jedoch eigenständig weiterentwickelt werden und das zur Verfügung stehende Entwicklungsteam hat Know-how in einer bestimmten Programmiersprache, nämlich Java. In diesem Fall ist es deshalb sinnvoll, die Verwendung von Java und, das darf nicht vergessen werden, die Auslieferung des ausreichend kommentierten Quellcodes als weitere Anforderungen zu formulieren. Eine solche Forderung ist darüber hinaus auch rechtlich zulässig, denn ein sachlicher Grund für die Forderung nach einer bestimmten Technologie ist hier gegeben. Zwar wäre eine Weiterentwicklung durch die Behörde grundsätzlich auch mit einer anderen Technologie möglich, dies wäre jedoch mit deutlich höheren Kosten verbunden, da die Mitarbeiter erst entsprechend geschult werden müssten.

---

1 SAGA V 3.0, www.kbst.bund.de, Berlin, Oktober 2006

2 Im nachfolgenden Beispiel 13 sieht das jedoch anders aus.

Beispiel 14 Die Behörde Z möchte ein neues Dokumenten-Management-System (DMS) einführen. Derzeit verwendet sie ausschließlich Oracle als Datenbank-Management-System (DBMS). Sie möchte sich jedoch die Möglichkeit offen halten, zukünftig auch andere Datenbank-Management-Systeme einzusetzen.

Ein gewünschtes Ziel für das DMS ist hier offensichtlich die Plattformunabhängigkeit bezüglich des DBMS. Das DMS muss mit verschiedenen Datenbank-Management-Systemen betrieben werden können. Und genau so sollte die entsprechende Anforderung auch formuliert werden, beispielsweise:

> *Das Dokumenten-Management-System muss mit verschiedenen Datenbank-Management-Systemen betrieben werden können, unter anderem Oracle, SQL-Server, PostgreSQL, MySQL, DB2 und INFORMIX. Die Portierung auf andere, hier nicht aufgelistete, Datenbank-Management-Systeme sollte allein durch entsprechende Umkonfiguration des DMS möglich sein.*

Bekanntlich werden relationale Datenbanken über die *Structured Query Language* (SQL) angesprochen. Diese Sprache wurde vom *American National Standards Institute* (ANSI) standardisiert. Der Standard SQL-92 wird von den allermeisten Datenbanksystemen umgesetzt.[1] Es spricht also grundsätzlich nichts dagegen, zusätzlich die ausschließliche Verwendung eines bestimmten Standard-SQLs, beispielsweise SQL-92, vorzuschreiben. Wichtig ist jedoch, dass diese Vorschrift allein die obige Anforderung nicht ersetzen kann. Viele DBMS-Produkte bieten neben der Standardsprache jedoch proprietäre Dialekte, d.h. Erweiterungen, an. Die Verwendung dieser Erweiterungen führt aber zu einer Abhängigkeit vom konkreten Produkt. Die Behörde könnte also auf die Idee kommen, die ausschließliche Verwendung eines bestimmten Standard-SQLs, beispielsweise SQL-92, vorzuschreiben. Ebenso wie im Beispiel 12 besteht hier jedoch die Gefahr, dass das eigentliche Ziel verfehlt wird, wenn sich die Behörde allein auf die Verwendung von Standard-SQL verlässt. Die Verwendung von Standard-SQL für die Kommunikation zwischen DMS und DBMS schließt nämlich nicht aus, dass andere proprietäre Eigenschaften eines DBMS-Produktes verwendet werden. Beispielsweise könnte das DMS eine spezielle Sprache anbieten, mit der Trigger definiert werden können, die beim Eintreten eines bestimmten Ereignisses, z.B. dem Einfügen eines neuen Dokuments, spezielle weitere Maßnahmen in der Datenbank auslösen. Die Definition solcher Trigger ist aber produktspezifisch. Beim Wechsel zu einem anderen DBMS müsste erstens sicher gestellt werden, dass das neue DBMS gleichwertige Funktionen anbietet,

[1] Aktuell ist der Standard SQL-2003, der aber noch nicht von allen Datenbanksystemen umgesetzt wird.

und zweitens müssten die Trigger für das neue DBMS neu geschrieben werden. Daraus wird deutlich, dass neben der Forderung nach der Verwendung von Standard-SQL gleichzeitig die Nichtverwendung von proprietären Funktionen zu fordern wäre.

Verfügt eine Organisation nicht über das nötige Maß an Kenntnissen und Erfahrungen oder sind diese nicht auf genügend Köpfe verteilt, um geeignete Spezifikationen erstellen zu können, sollte die Entwicklung einer Spezifikation zusammen mit einem externen Projektpartner vorgenommen werden.

Das Ziel der Spezifikationstreue wird dadurch erreicht, dass der Herstellungsprozess ein brauchbares Ergebnis hervorbringt. Das betrifft vor allem die Entwicklung von Fachanwendungen, denn beim Kauf gibt es keine Möglichkeit, Einfluss auf den Herstellungsprozess zu nehmen. Dieser Aspekt wird daher in Abschnitt 4.2.1.2 näher betrachtet.

Nachgewiesen wird Spezifikationstreue dadurch, dass das Ergebnis mit geeigneten Verfahren auf Einhaltung der Spezifikation geprüft wird. Dies ist sowohl beim Kauf als auch bei der Entwicklung möglich. Allerdings gestaltet sich die konkrete Ausführung im Detail etwas unterschiedlich, wie in den Abschnitten 4.2.1.2 und 4.2.1.3 erläutert wird.

Diese drei Aufgaben, Spezifikation, Prozessgestaltung und Verifikation müssen im Zuge der Beschaffung einer Fachanwendung bewältigt werden. Ihre Gewichtung hängt allerdings davon ab, ob die Anwendung mehr oder weniger fertig als Standardsoftware oder als Individualsoftware beschafft werden soll. Bei der Beschaffung als Individualsoftware muss nochmals unterschieden werden zwischen einer Auftragsentwicklung und einer Eigenentwicklung[1].

Die Frage, ob eine Fachanwendung 'off the shelf' gekauft[2], intern oder aber extern entwickelt wird, stellt sich häufig nicht in dieser Vielfalt. Manche Fachanwendung steht in keinem Regal ("shelf"). Manche Organisation hat keine Kapazitäten für Eigenentwicklungen. Wenn sich aber tatsächlich alle drei Optionen anbieten, wie kann der Pfad zur Entscheidung für eine dieser Optionen strukturiert werden? Wie kann eine IT-Gesamtarchitektur diese Entscheidung unterstützen? Wichtige Hinweise dazu liefern die Betrachtung der verschiedenen Aufgaben beim Kauf und bei der Entwicklung und ein kritischer Blick darauf, wie die Organisation für diese Aufgaben gerüstet ist oder sich rüsten kann.

[1] Im Dienste der Klarheit orientieren sich die folgenden Ausführungen an diesen drei Alternativen. Allerdings wäre eine weitaus feinere Differenzierung möglich. Vorstellbar sind beispielsweise interne Dienstleistungsbeziehungen. Es sollte leicht möglich sein, die im Folgenden dargestellten Überlegungen auf derartige Szenarien anzuwenden.

[2] Als Standardsoftware kann man sie ggf. auch mieten, leasen oder über einen ASP beschaffen.

4.2.1.2 Entwicklung

Die Entwicklung einer individuellen Fachanwendung unterscheidet sich von der Beschaffung einer Standardanwendung, sei es nun per Kauf, Miete, Leasing oder über einen Application Service Provider (ASP), vor allem dadurch, dass Einfluss auf den Entwicklungsprozess genommen werden kann. Diese Einflussnahmemöglichkeiten können dazu genutzt werden, einer speziellen Form der Plattformabhängigkeit zu begegnen, nämlich der Abhängigkeit von der Technologieplattform. Letztendlich ist jede Anwendung mit Hilfe irgendeiner Technologie implementiert, wodurch eine gewisse Abhängigkeit von dieser Technologie entsteht. Diese Abhängigkeit äußert sich auf zweierlei Weise:

- Die Integration mit Anwendungen, die mit Hilfe einer anderen Technologie implementiert wurden, wird erschwert, da technologiespezifische Schnittstellen und Protokolle nicht verwendet werden können.
- Der Wechsel zu einer anderen Technologieplattform wird erschwert. Eine in Java implementierte Anwendung lässt sich beispielsweise nicht ohne weiteres auf die .NET-Plattform portieren und umgekehrt.

Das erste Problem kann durch die Wahl einer geeigneten Systemarchitektur gelöst werden. Wie in Abschnitt 4.1.3.4 dargelegt wurde, können mit Hilfe einer serviceorientierten Architektur Anwendungen integriert werden, die mittels unterschiedlichster Technologien implementiert wurden.

Dem zweiten Problem kann durch ein modellgetriebenes Vorgehen entgegen getreten werden. Das Grundprinzip eines solchen Vorgehens ist die Definition einer Reihe von Modellen, die auseinander ableitbar sind. Der Unified Software Development Process (als "Rational Unified Process" in ein Werkzeug gegossen) schlägt beispielsweise ein Use Case Model vor, ferner ein Analysis-, Design-, Implementation-, Deployment- sowie ein Test Model. Das V-Modell[1] folgt einer ähnlichen Idee.

Der Nutzen dieser Modell-Ketten liegt aus Sicht der Plattformunabhängigkeit darin, dass das Software-System auf einer Reihe unterschiedlicher Abstraktionsebenen definiert ist. Die abstrakteste dieser Ebenen enthält eine geschäftslogische – und damit technologieunabhängige – Beschreibung des Software-Systems. Die konkreteste Ebene, die Implementation, ist dagegen durch und durch plattformabhängig. Soll eine so entwickelte Anwendung auf eine andere Plattform transferiert werden, wird zwar die konkrete Implementation obsolet. Die abstrakteren Modelle behalten jedoch ihren Wert, mit ihnen die eigentliche intellektuelle Leistung, und damit das eigentliche Investment. Die konsequenteste Form dieses Vorgehens ist die *modellgetriebene Softwareentwicklung* (Model-driven Software Development, MDSD). Bei dieser Vorgehensweise werden die

[1] siehe http://www.kbst.bund.de/-,279/V-Modell.htm

Modelle mit Hilfe formalisierter Sprachen, so genannter *domänenspezifischer Sprachen* (Domain-specifc Language, DSL) formuliert. Die konkrete Implementierung, d.h. die Transformation auf eine konkrete Technologieplattform, erfolgt dann automatisiert mit Hilfe von Werkzeugen. Um eine auf diese Weise entwickelte Anwendung auf eine andere Technologieplattform zu portieren, muss lediglich die automatisierte Generierung für die andere Plattform ausgeführt werden.

Ein weiterer Nutzen von Modellen ist ihre Abhängigkeit voneinander. Aneinandergereiht markieren sie einen Weg von der Anforderung zur Validierung des fertigen Systems. Die Anforderungsgerechtigkeit des Systems befördern sie dadurch, dass sie die Abbildungen der Modelle aufeinander verfolgbar machen. Bei konsequenter Anwendung der Methode ist es jederzeit für einen gegebenen Anwendungsfall möglich zu identifizieren, welche Software-Komponenten auf welche Weise interagieren, um ihn zu realisieren, und durch welche Testfälle er verifiziert wird.

Die Einflussnahmemöglichkeiten auf den Entwicklungsprozess können auch dazu genutzt werden, die Verifikation des Ergebnisses zu erleichtern. Einige Merkmale des Ergebnisses können bereits durch die Beschaffenheit des Prozesses gewährleistet werden. Ist beispielsweise eine der Anforderungen, dass die zu entwickelnde Anwendung auf unterschiedlichen Betriebssystemen lauffähig ist und dass die von der Anwendung zur Verfügung gestellte grafische Benutzeroberfläche auf all diesen Betriebssystemen gleich aussieht, kann der Entwicklungsprozess vorsehen, dass entwicklungsbegleitende Tests kontinuierlich auf unterschiedlichen Betriebssystemen durchgeführt werden. Soll sichergestellt werden, dass bestimmte Architekturmuster verwendet werden, kann der Entwicklungsprozess vorsehen, dass Architekturentscheidungen, Schnittstellenspezifikationen etc. durch ein Gremium getroffen werden, in dem auch im Falle der Auftragsentwicklung die Behörde vertreten ist, möglicherweise mit Veto-Recht.

Merkmale, die bereits durch die Beschaffenheit des Prozesses gewährleistet werden können, sollten zwar nach wie vor verifiziert werden, das kann aber in vielen Fällen bereits durch den Nachweis geschehen, dass der Prozess in seinen jeweils maßgeblichen Teilen tatsächlich befolgt wurde. Der Aufwand für die Spezifikation, Vorbereitung und Durchführung technischer Konformitätstests kann dann entfallen. Die Bedeutung der Verifikation verhält sich insofern umgekehrt proportional zur Prozesskontrolle.

Offenkundig wird für die Eigenentwicklung von Fachanwendungen Personal mit entsprechendem Know-how benötigt. Wie die obigen Ausführungen zeigen, sind Kenntnisse in Methoden der Softwareentwicklung jedoch nicht nur für Organisationen unverzichtbar, die selbst Softwareentwicklung betreiben. Im Rahmen von Auftragsentwicklungen kann sie erhebliche Gestaltungsspielräume eröffnen, nicht zuletzt im Dienste der Plattformunabhängigkeit.

Die Konsequenzen sind unmittelbar ersichtlich: Je tiefer die Organisation in die Gestaltung des Prozesses eingreifen oder in Entscheidungen eingebunden werden will, desto mehr muss sie geeignetes Personal bereitstellen. Und Eignung bedeutet in diesem Zusammenhang eine ausreichende Kenntnisse in Technologie-, Entwurfs-, Prozess- und Werkzeugfragen, die der des Dienstleisters nicht wesentlich nachsteht. Der Aufbau von Methodenwissen in diesem Themenfeld ist im Rahmen eines strategischen Abhängigkeitsmanagements in jedem Falle eine lohnende Investition.

4.2.1.3 Beschaffung von Standardsoftware

Die Beschaffung einer existierenden Standard-Anwendung bietet im Regelfall keinerlei Einflussmöglichkeiten auf den Entwicklungsprozess. Folglich kommt hier der Verifikation der Anforderungen eine besondere Bedeutung zu, denn nur durch die Verifikation kann sichergestellt werden, dass die Anwendung tatsächlich alle geforderten Eigenschaften hat.

Üblicherweise geht dem Kauf eine Marktanalyse voran, in der anhand der in der Spezifikation niedergelegten Anforderungen verifiziert wird, ob der Markt entsprechende Lösungen bietet. Die Verifikation von Eigenschaften erfolgt hier also in Form einer Evaluation. Für die Fachanwendung aus Beispiel 12 müssten beispielsweise geprüft werden, ob es Lösungen am Markt gibt, die auf unterschiedlichen Betriebssystemen genutzt werden können. Für das DMS aus Beispiel 14 wäre u.a. zu prüfen, ob es Lösungen am Markt gibt, die mit unterschiedlichen Datenbank-Management-Systemen betrieben werden können. In der der Marktanalyse folgenden produktunabhängigen Ausschreibung werden dann die verschiedenen Anforderungen zu Bestandteilen des Pflichtenhefts, ähnlich wie bei der Auftragsentwicklung.

Wie die Beispiele zeigen, wird auch beim Kauf ein gewisses Maß an einschlägigem technischen Know-how benötigt, da auch nichtfunktionale Anforderungen zu spezifizieren und prüfen sind. In jedem Fall sollte so viel Wissen um Softwaretechnologien und -architekturen in den Beschaffungsprozess einfließen, dass die neue Fachanwendung mit der strategisch angestrebten Anwendungslandschaft und IT-Gesamtarchitektur harmonisiert werden kann. Der Kauf einer fertigen Anwendung kann so bereits technisch in die gewünschte Richtung gelenkt werden.

Darüber hinaus wird eine entsprechende IT-Infrastruktur benötigt, in der die Evaluationen durchgeführt werden können. Ist diese nicht vorhanden oder fehlt das benötigte technische Know-how bzw. Personal, sollte, wie schon bei der Spezifikationserstellung, die Hilfe eines externen Dienstleisters in Betracht gezogen werden.

Da der Kauf einer bereits existierenden Anwendung i.d.R. im Rahmen einer Ausschreibung erfolgt, besteht im Hinblick auf die Bewertung der angebotenen Produkte auch die Möglichkeit, die geforderten Kriterien zur Plattformunabhängigkeit

im Rahmen eines so genannten „Proof of Concept“ (PoC) durch den Anbieter nachweisen zu lassen. Dabei muss aber auch immer die Verhältnismäßigkeit der Mittel berücksichtigt werden. Auch die Anbieter bewegen sich bei der Abgabe von Angeboten im Rahmen der Wirtschaftlichkeit. Wird also für den Anbieter das Angebot teurer als der Gewinn aus dem Verkauf seines Produktes, so wird er vermutlich kein Angebot abgeben. Wenn also die Aufwendungen für einen PoC nicht in angemessener Relation zum Gesamtaufwand stehen, sollte darauf verzichtet werden, weil es sonst passieren kann, dass keine Angebote abgegeben werden. Ausnahme: Unter Umständen haben die Bieter einen Anspruch auf Erstattung der Kosten z.B. aus § 29 Nr. Abs. 1 VOL/A. Wenn die beschaffende Stelle die Kosten für einen PoC erstattet, dann kann sie auch von jedem Bieter einen PoC verlangen. Aber auch hier gilt der Grundsatz der Wirtschaftlichkeit.

4.2.1.4 Bewertung der angebotenen Leistungen zur Plattformunabhängigkeit

Grundsätzlich fließen die im Rahmen einer Ausschreibung angebotenen Leistungen zur Plattformunabhängigkeit nicht anders in die Bewertung ein als alle anderen Leistungen. Die UfAB[1] bietet den Behörden hierzu eine Hilfestellung und Methodik, mit der es der Behörde zum einen möglich ist, die spezifizierten Anforderungen an die Fachanwendung und den Lieferanten in einen entsprechenden Kriterienkatalog mit entsprechender Gewichtung der einzelnen Kriterien zu überführen und zum anderen eine Bewertung der angebotenen Leistungen durchzuführen, die dem Gleichbehandlungsgrundsatz gerecht wird.

Die UfAB sieht für solch einen Kriterienkatalog eine differenzierte Gruppierung der Kriterien in unterschiedliche Kriteriengruppen vor. Diese Gruppierung kann gem. UfAB auch auf mehreren Ebenen z.B. Kriterienhauptgruppen, Kriteriengruppen, Einzelkriterien erfolgen. Die Komplexität des Kriterienkataloges hängt letztendlich auch von der Komplexität der ausgeschriebenen Leistung ab. Die einzelnen Kriteriengruppen erhalten je nach ihrer Wichtigkeit für das abzuliefernde Gesamtergebnis, im vorliegenden Fall die zu beschaffenden Fachanwendung, eine entsprechende Gewichtung[2] Der Grad der Detaillierung innerhalb einer Kriteriengruppe soll sich dabei gem. UfAB auch an der Gewichtung der jeweiligen Gruppe orientieren.

Was bedeutet dies nun für die Anforderungen an eine Fachanwendung bezüglich der Plattformunabhängigkeit? Zunächst einmal ist festzuhalten, dass die Kriterien zur Plattformunabhängigkeit nie die Gewichtung der Kriterien erreichen oder gar überschreiten können, die sich unmittelbar aus der Fachaufgabe ableiten lassen. Mit einer Fachanwendung sollen fachliche Aufgaben erledigt werden, daher wird

[1] siehe http://www.kbst.bund.de/ „Wirtschaftlichkeit und Recht / UfAB

[2] siehe http://www.kbst.bund.de/ „Wirtschaftlichkeit und Recht / UfAB

die Gesamtheit der aufgabenbezogenen Kriterien immer die höchste Gewichtung innerhalb des Kriterienkataloges einnehmen.

Die Kriterien zur Plattformunabhängigkeit könnten natürlich auch ohne eine besondere Betrachtung z.B. in den anderen technischen Kriterien mit bedacht werden. Hat aber die Plattformunabhängigkeit für die Realisierung der Fachanwendung eine gewisse Bedeutung, sei aus rein technischer Sicht oder aber auch aus strategischer Sicht, dann ist es durchaus sinnvoll für die Plattformunabhängigkeit eine eigene Kriteriengruppe mit entsprechender Gewichtung zu definieren. Innerhalb dieser Kriteriengruppe werden dann die Ziele und Anforderungen, welche die Plattformunabhängigkeit in der gewünschten Art und Weise sicherstellen sollen, so wie in den vorangegangen Abschnitten erläutert, als Kriterien formuliert und wiederum je nach ihrer Wichtigkeit mit Punkten bedacht.

Die Festlegung der Einzelkriterien kann sich dabei auf die Definition der Ziele, wie z.B. „ … muss auf beliebigen Betriebssystem, mindestens aber auf …lauffähig sein.“ oder „… muss mit beliebigen Office-Anwendungen, mindestens aber … zusammen arbeiten.“ beschränken aber auch, wenn dies sachlich gerechtfertigt ist, Technologien, konkrete Standards usw. berücksichtigen.

Um die Erfüllung der geforderten Ziele ausreichend gut bewerten zu können, muss vom Anbieter gefordert werden, darzulegen, welchen (technischen) Weg er vorsieht, um das jeweilige Ziel umzusetzen. Dabei muss dem Anbieter vorgegeben werden, wie detailliert und ggf. in welcher Form dieser Weg zu beschreiben ist. Zum einen können dabei die in Kapitel 3.1 dargelegte Definition und Differenzierung von Plattformunabhängigkeit herangezogen werden, also wie sollen die jeweiligen Schichten realisiert werden und zum anderen kann auch die Beschreibung von eingesetzten Schnittstellen und Standards verlangt werden. Alternativ dazu kann, wie bereits im vorangegangenen Kapitel dargelegt, ein PoC gefordert werden.

Kriterien, die bestimmte Technologien oder Standards abfragen, können häufig nur mit ja oder nein beantwortet werden, eine feinere Differenzierung ist meist nicht möglich. Dies ist zwar nicht ganz im Sinne der UfAB, steht dem aber auch nicht entgegen, wenn es sich nicht nur um solche Kriterien handelt. Es sei denn, es handelt sich um so genannte A-Kriterien (Ausschluss-Kriterien). In diesem Fall ist ein Kriterium, welches in seiner Antwort nur ja oder nein zulässt, ein optimales Kriterium.

Wichtig ist aber auch, dass geprüft wird, ob das Personal des Anbieters, das notwendige Know-how für die jeweiligen Technologien, Standards usw. mitbringt.

Damit bietet sich den Behörden eine Methodik und ein Werkzeug, mit dem schon im Rahmen der Ausschreibung das notwendige Maß an Plattformunabhängigkeit sicher gestellt werden kann.

4.2.2 Wirtschaftliche Aspekte

In der Diskussion um Plattformunabhängigkeit oder Abhängigkeiten ganz allgemein wird immer wieder die Frage aufgeworfen: „Ist die Beschaffung von plattformabhängigen Fachanwendungen nicht günstiger als die von plattformunabhängigen?“

Auf den ersten Blick kann das in vielen Fällen durchaus der Fall sein. Insbesondere, wenn man nur die reinen Beschaffungskosten, also Entwicklungs- oder Kaufkosten und Implementierungskosten betrachtet. Wie aber bereits im Kapitel 1 erwähnt, sind Abhängigkeiten immer dann problematisch, wenn sie Veränderungen behindern. Die Wahrscheinlichkeit, dass Veränderungen vorgenommen werden müssen, sei es an der Fachanwendung selbst oder an ihrer Ablaufumgebung, steigt mit zunehmender Einsatzdauer. Daran wird deutlich, dass für die Betrachtung der Wirtschaftlichkeit einer Lösung der Betrachtungszeitraum eine sehr wesentliche Rolle spielt. Unabhängigkeit und Flexibilität schaffen langfristig mehr Wirtschaftlichkeit. Im Vorfeld der Wirtschaftlichkeitsbetrachtung muss daher genau geprüft werden, für welchen Zeitraum die Fachanwendung eingesetzt werden soll und welche Veränderungen wann zu erwarten sind. Dabei sind Veränderungen nicht immer direkt zu identifizieren. Wie uns die Erfahrung gelehrt hat, kann der Wunsch nach Veränderung, z.B. des Betriebssystems, um mehr Herstellerunabhängigkeit zu erhalten, relativ kurzfristig entstehen. So wurde in der Vergangenheit nur selten bei der Beschaffung einer Fachanwendung daran gedacht, dass diese in der Zukunft vielleicht auch auf einem anderen Betriebssystem laufen soll als dem für welches sie beschafft wurde. Wenn nun Überlegungen bezüglich eines Wechsels des Betriebssystems gemacht werden, kommt man häufig zu dem Ergebnis, dass die Abhängigkeit der Fachanwendung vom alten Betriebssystem diese Veränderung behindert und ihre Wirtschaftlichkeit gar in Frage stellt. Dies gilt aber nicht nur für die Abhängigkeit von Betriebssystemen sondern auch bei der Abhängigkeit von Standardanwendungen, wie z.B. Office-Anwendungen, Abhängigkeiten von Datenbank-Management-Systemen oder bei Abhängigkeiten zwischen Fachanwendungen. Selbst wenn eine Fachanwendung in sich sehr starr, aufgebaut ist, also eine starke innere Abhängigkeit besitzt, kann dies zu Problemen führen. Fachanwendungen helfen häufig gesetzliche Vorschriften umzusetzen, müssen sich also an die Gestaltung der Gesetze anpassen lassen. Wenn nun solch eine gesetzliche Vorschrift häufig verändert wird, kann ein zu starrer Programmaufbau der Fachanwendung zu erheblichen Anpassungskosten führen und damit die Wirtschaftlichkeit der gewählten Lösung in Frage stellen. Daraus wird ersichtlich, dass die Beurteilung der Wirtschaftlichkeit einer Lösung für eine zu beschaffende Fachanwendung auch sehr stark von den Anforderungen hinsichtlich Einsatzdauer und Anpassungsfähigkeit an die jeweilige Fachanwendung und die sie umgebende Ablaufumgebung abhängig ist. Grundsätzlich lässt sich aber festhalten, dass mehr Flexibilität und mehr Unabhängigkeit langfristig auch mehr Wirtschaftlichkeit gewährleisten. Daher sollte Plattformunabhängigkeit aus wirtschaftlicher Sicht eine wesentliches Kriterium für

die Beschaffung von Fachanwendungen sein. Letztendlich muss aber die Wirtschaftlichkeitsbetrachtung aufzeigen, ob es nicht doch plattformabhängige Lösungen gibt, die wirtschaftlicher sind.

In berechtigten Fällen kann eine Behörde auch eingeschränkte Plattformunabhängigkeit oder sogar Plattformabhängigkeit zulassen. Die berechtigten Fälle definieren sich dann durch die Wirtschaftlichkeit.

Die Wirtschaftlichkeitsbetrachtung muss den Nachweis erbringen, dass

- eingeschränkte Plattformunabhängigkeit: (siehe Kapitel 3.1)
  - langfristig in die Plattformunabhängigkeit überführt werden kann oder
  - langfristig günstiger ist als Plattformunabhängigkeit oder
  - der Einsatz der problematischen Software einen temporären Charakter hat (Ersatzmaßnahme),
- Plattformabhängigkeit (siehe Kapitel 3.1)
  - sowohl kurz- als auch langfristig günstiger ist oder
  - der Einsatz der problematischen Software einen temporären Charakter hat.

4.2.2.1 Vorgehensweise in der Wirtschaftlichkeitsbetrachtung

Die Vorgehensweise in der Wirtschaftlichkeitsbetrachtung soll sich in jedem Fall nach § 7 BHO richten (siehe Abschnitt, „Grundsätze“, S. 112). Dies gilt insbesondere für Maßnahmen zur Erstellung, Überarbeitung oder Beschaffung von IT-Fachanwendungen.

Der eigentlichen Wirtschaftlichkeitsbetrachtung der Maßnahme zur Erstellung bzw. Modifizierung einer IT-Fachanwendung sollten sog. "Vorfeldbetrachtungen" vorausgehen (siehe Abschnitt „Vorfeldbetrachtungen“, S. 114). In diesem Schritt werden der tatsächliche Handlungsbedarf sowie mögliche Realisationsalternativen verifiziert.

Ist der grobe Rahmen abgesteckt, so wird eine detaillierte Projektplanung erforderlich, die sich an dem V-Modell[1] orientieren soll. Diese Projektplanung ist das Kernstück und die Basis für die Wirtschaftlichkeitsbetrachtung der Maßnahme.

Als Methodik der Wirtschaftlichkeitsbetrachtung empfiehlt sich die WiBe 4.0. Diese Methodik berücksichtigt neben dem "quantitativen" Faktor der Kennzahl für Kosten/Nutzen auch "qualitative" Faktoren für Dringlichkeit und Qualität sowie die

[1] V-Modell: Vorgehensmodell zur Planung und Durchführung von IT-Vorhaben, Entwicklungsstandard für IT-Systeme des Bundes, V-Modell XT Release 1.2, hrsg. von der KBSt, Koordinierungs- und Beratungsstelle der Bundesregierung für Informationstechnik in der Bundesverwaltung 2006 (www.kbst.bund.de)

Perspektive des Kunden, die mit der Kennzahl für externe Effekte aus Kundensicht berücksichtigt wird.

Grundsätze

Kapitel 2.1 der Verwaltungsvorschrift zu § 7 Bundeshaushaltsordnung (BHO) besagt: "Wirtschaftlichkeitsuntersuchungen in der Planungsphase bilden die Grundlage für die begleitenden und abschließenden Erfolgskontrollen. Wirtschaftlichkeitsuntersuchungen müssen mindestens Aussagen zu folgenden Teilaspekten enthalten:

- Analyse der Ausgangslage und des Handlungsbedarfs,
- Ziele, Prioritätsvorstellungen und mögliche Zielkonflikte,
- relevante Lösungsmöglichkeiten und deren Nutzen und Kosten (einschl. Folgekosten), auch soweit sie nicht in Geld auszudrücken sind,
- finanzielle Auswirkungen auf den Haushalt,
- Eignung der einzelnen Lösungsmöglichkeiten zur Erreichung der Ziele unter Einbeziehung der rechtlichen, organisatorischen und personellen Rahmenbedingungen,
- Zeitplan für die Durchführung der Maßnahme,
- Kriterien und Verfahren für Erfolgskontrollen (vgl. Nr. 2.2, VV zu § 7 BHO)."

Diese Anforderungen bilden prinzipiell den Rahmen und die Struktur für die Wirtschaftlichkeitsbetrachtungen (vgl. Abbildung 24, S. 113). Ein wesentlicher Bestandteil sind die Zielvorstellungen der Behörde, in diesem Zusammenhang auch mögliche Zielkonflikte sowie Rahmenbedingungen in Form von Vorgaben und Annahmen. Die Abstimmung des Modells der Wirtschaftlichkeitsbetrachtung gehört auch in diesen Bereich. Daneben ist die Ist-Situation zu erheben, wobei Informationen zur Infrastruktur, der Hard- und Softwareprodukte, der anderen Fachanwendungen u.v.m. ermittelt werden. Auf dieser Basis werden technische Lösungsmöglichkeiten eruiert und mit Kostenansätzen unterlegt. Dazu gehören neben den Kosten für Hard- und Software auch die Aufwendungen für externes und internes Personal. Die sich nun anschließende Ermittlung der Wirtschaftlichkeit berechnet Kosten und Nutzen der IT-Maßnahme und die Auswirkungen auf den Haushalt. Zur Einschätzung der Personalkosten wird eine grobe Projektplanung erforderlich, die auch einen Zeitplan für die Durchführung der IT-Maßnahme beinhaltet.

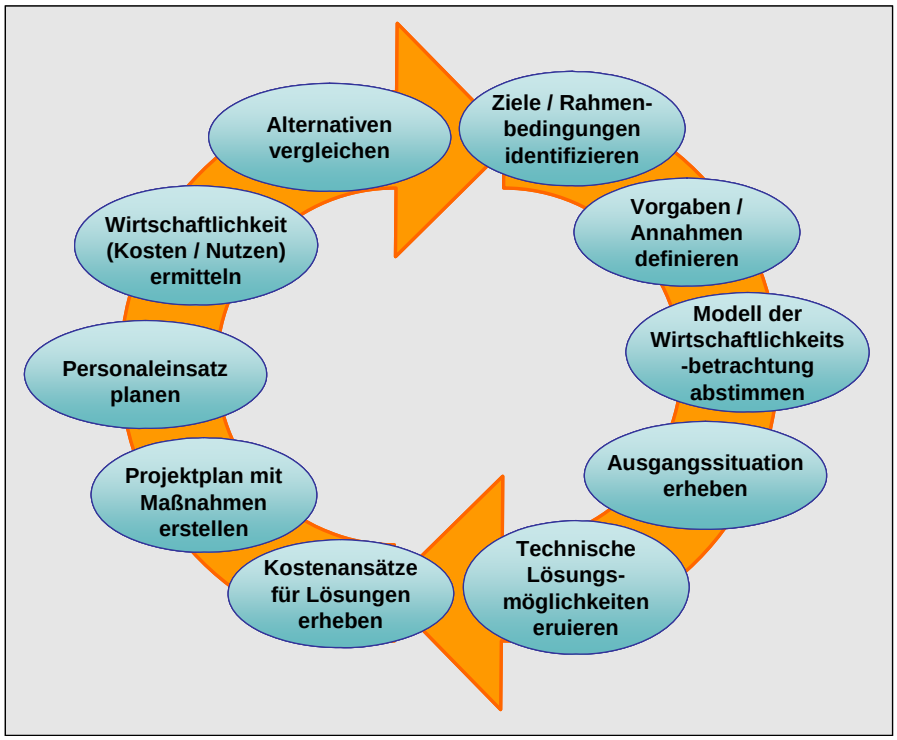


Abbildung 24: Regelkreis der Wirtschaftlichkeitsbetrachtung

Vor der Durchführung einer jeden IT-Maßnahme sollten die im IT-Rahmenkonzept festgelegten oder aus strategischen Vorgaben abgeleiteten operativen Behördenziele festgestellt werden. Ein Abgleich dieser Ziele mit der IT-Maßnahme reduziert mögliche Konflikte mit den Zielvorgaben und anderen Projekten. Weiterhin wird hiermit der Grundstein für die in Kapitel 2.2 der VV zu § 7 BHO geforderten Zielerreichungskontrolle gelegt.

Die Erstellung eines grundsätzlichen Ziel- und Anforderungssystems für die IT-Maßnahme hilft, mögliche Lösungen unabhängig von der Wirtschaftlichkeit auf ihre Eignung hin zu prüfen. Hiermit wird konkret die Definition von Anforderungskriterien angesprochen, die für die einzelnen Lösungsalternativen in Form von Nutzwertanalysen[1] zu bewerten sind. Einzelne dieser Kriterien oder Kriteriengruppen können im Rahmen der Zielerreichungskontrolle als Messgröße dienen.

An dieser Stelle sei der Hinweis gegeben, dass die Definition der Zielsetzungen der Behörde und die damit verbundenen notwendigen Aktivitäten in einem separaten Strategieprozess (siehe Kapitel 4.1) gefunden werden sollten.

[1] Vgl. WiBe 4.0, Empfehlung zur Durchführung von Wirtschaftlichkeitsbetrachtungen in der Bundesverwaltung, insbesondere beim Einsatz der IT, Version 4.0 – 2004, "Berechnung der erweiterten Wirtschaftlichkeit", S. 80 ff

Vorfeldbetrachtungen

Rahmenbedingungen

Diese Informationen helfen, den Rahmen abzustecken und gewisse Aktivitäten schon im Vorfeld entsprechend zu kanalisieren (z.B. Prozessanalyse in eigenständigem Projekt). Auch ist hier die IT-Strategie der Behörde zu berücksichtigen, die entsprechende Vorgaben oder Restriktionen, wie z.B. das Ziel der Plattformunabhängigkeit oder der Herstellerunabhängigkeit, beinhalten kann. Für die Fachanwendungen stellen hier die IT-Gesamtarchitektur sowie die Architekturrichtlinien die ausschlaggebenden Produkte der IT-Strategie dar, die zu berücksichtigen sind.

Ist-Situation

Hiermit wird die Ausgangssituation für die Maßnahme beurteilt. Anhand der dort aufgeführten Fragen kann ein konkreter Handlungsbedarf abgeleitet werden, der in eine Maßnahme münden kann, für die dann eine WiBe zu erstellen ist. Mit der kommentierten und dokumentierten Beantwortung dieser Fragen wird eine solide Basis für die Begründung der Maßnahme gelegt (Beantwortung der Frage: "Welches Problem soll mit der Beschaffung der Fachanwendung gelöst werden?").

Soll-Szenario

Mit diesen Fragestellungen kann die Sinnhaftigkeit der Beschaffung der Fachanwendung schon im Vorfeld geprüft werden. Die fokussierten Alternativen sind hinreichend zu kommentieren und begründen. Damit entsteht eine Dokumentation, die wiederum die Grundlage für die Begründung der Realisierungsalternative liefert (Beantwortung der Frage; "Warum und wofür soll die Fachanwendung beschafft werden?").

Grundsätzliche Überlegungen zur Kostenerhebung

Hier werden im Vorfeld der eigentlichen Projektplanung Informationen gegeben zu möglichen Projektphasen, dem Personalbedarf sowie der Eruierung der Soft- und Hardwarekosten. Damit sollen auch schon in dieser Phase grobe Kostenschätzungen in die Beurteilung zur Priorisierung der Maßnahme einfließen.

Hier sei nochmals darauf verwiesen, dass IT-Maßnahmen unter dem Gesichtspunkt der Plattformunabhängigkeit in der gleichen Weise einer Wirtschaftlichkeitsbetrachtung zu unterziehen sind, wie andere Projekte.

Die zu erwartenden Kosten-Einsparungen sind in der Regel schwierig zu quantifizieren. Soweit dies aber möglich ist, sollte ein Kostenansatz für Ersparnisse in die Wirtschaftlichkeitsbetrachtung aufgenommen werden. Mögliche Einspareffekte ergeben sich zum Beispiel

- bei absehbaren Plattformwechseln in der Zukunft (Vermeidung von Neuanschaffungen, Einsatz wirtschaftlicherer Plattformen),
- bei der Wiederverwendbarkeit von Fachanwendungen auf unterschiedlichen Plattformen (Vermeidung von Mehrfachentwicklung oder –kauf).

An dieser Stelle ist zu überlegen, ob eventuell auch Nebeneffekte durch andere Projekte, auch nur teilweise, berücksichtigt werden können.

Plattformunabhängigkeit und der Betrachtungszeitraum

Da Plattformunabhängigkeit einen eher (mittel- bis) langfristigen strategischen Ansatz verfolgt, muss die Wirtschaftlichkeitsbetrachtung dem folgen. Mit plattformunabhängigen Fachanwendungen kann der Produktlebenszuklus verlängert werden. Eine Überarbeitung der Fachanwendung wird nun nicht mehr durch die alleinige Umstellung der Plattform, auf der diese Anwendung laufen muss, notwendig.

Da es sich bei der Plattformunabhängigkeit um langfristige Zielstellungen handelt, sollte der Betrachtungszeitraum einer WiBe, wie für solche Fälle in der WiBe 4.0[1] empfohlen, eine größere Zeitspanne als 5 Haushaltsjahre umfassen. Hier ist von einer bis zu 10-jährigen Betrachtungszeit auszugehen. Deren tatsächliche Dauer sollte aber in jedem Projekt individuell an die Bedürfnisse bzgl. Plattformunabhängigkeit angepasst werden. So wird in der Regel eine Fachanwendung, die *keine Plattformunabhängigkeit* (siehe Tabelle 1) realisiert, einen kürzeren Lebenszyklus aufweisen als eine Fachanwendung die eine *Plattformunabhängigkeit* aufweist (siehe Tabelle 1).

Fachkonzept

Die Ermittlung der Wirtschaftlichkeit einer Maßnahme zur Erstellung oder Modifizierung einer IT-Fachanwendung ist abhängig von der Projektplanung. Die darin enthaltenen Personalaufwende sowie entsprechend spezifizierte Zusatz-Erfordernisse (Teststellungen, spezielle Software, externe Unterstützung, Schulung, Training, etc.) bilden die Basis für die Berechnungen. Die Projektplanung soll in ein "Fachkonzept" eingebunden werden, das zum einen die Notwendigkeiten in Bezug auf den unterstützen Prozess (die Prozesse) in Form einer Anforderungsanalyse darstellt und zum anderen ein Detailkonzept für die Soll-Sicht liefert.

Die Anforderungsanalyse kann klassischerweise in Form eines Anforderungskataloges erstellt werden. Die Kriterien eines solchen Kataloges sollen nachfolgende Bereiche abdecken:

[1] WiBe 4.0, Empfehlung zur Durchführung von Wirtschaftlichkeitsbetrachtungen in der Bundesverwaltung, insbesondere beim Einsatz der IT, Version 4.0, Schriftenreihe der KBSt, ISSN 0179-7263, Band 68, August 2004

- Fachliche Anforderungen
- Technische Anforderungen
- Anforderungen an die IT-Sicherheit
- Qualitative Anforderungen
- Sonstige Anforderungen

Das Detailkonzept soll sich an dem V-Modell XT orientieren[1].

Ermittlung der WiBe

Monetäre Wirtschaftlichkeit

Hier erfolgt nun die eigentliche Berechnung der Wirtschaftlichkeit. Im monetären Bereich werden für die Phasen Entwicklung bzw. Einführung (also die IT-Maßnahme selbst) und Betrieb Kennzahlen für Kosten und Nutzen ermittelt. Stehen dabei mehrere Alternativen zur Verfügung, z.B. eine plattformabhängige und eine plattformunabhängige Lösung, dann erfolgt die Ermittlung der Kennzahl für jede dieser Alternativen. Eine positive Kennzahl an dieser Stelle ergibt schon einen sehr hohen Befürwortungsfaktor für die jeweilige IT-Maßnahme. Aus monetärer Sicht wird die wirtschaftlichste der betrachteten Alternativen dadurch ermittelt, dass die gewonnen Ergebnisse gegenübergestellt werden.

Zur Erstellung der Wirtschaftlichkeitsbetrachtung soll die Methodik der WiBe 4.0[2] der KBSt zugrunde gelegt werden. Danach werden Kosten und Nutzen für die Bereiche Entwicklung und Betrieb der IT-Maßnahme erhoben. Der Betrachtungszeitraum soll sich dabei an der realen Nutzungsdauer der IT-Fachanwendung ausrichten.

Eine Risikobetrachtung der erhobenen Kosten/ Nutzen kann optional ergänzt werden.

[1] siehe www.kbst.bund.de, Standards und Architekturen / V-Modell

[2] siehe www.kbst.bund.de , Wirtschaftlichkeit und Recht / Wirtschaftlichkeit / Dokumente

Erweiterte Wirtschaftlichkeit

Mit der erweiterten Wirtschaftlichkeit werden qualitative Kriterien bewertet. Dies geschieht mit Hilfe der Nutzwertanalyse. Auch hier ist die Methodik der WiBe 4.0[1] der KBSt anzuwenden. Darin werden Kennzahlen ermittelt für die Kriterienbereiche:

- Dringlichkeit[2]
- Qualität und Strategie[3]
- Externe Effekte aus Kundensicht[4]

Um der besonderen Bedeutung der Plattformunabhängigkeit für die Zukunft gerecht zu werden, wurde das Kriterium Herstellerunabhängigkeit[5] um die Plattformunabhängigkeit in der WiBe 4.0 erweitert. Die neue Beschreibung des Kriteriums ist unter anderem im Anhang (siehe A.1) und auf der Webseite der KBSt (www.kbst.bund.de) dokumentiert.

Zur Punktbewertung des neuen Kriteriums Plattform-/Herstellerunabhängigkeit bezogen auf eine bestimmte Fachanwendung steht die nachfolgende Notenskala, Tabelle 3, zur Verfügung. Die Bewertung erfolgt hinsichtlich der Qualität der Plattformunabhängigkeit einer konkreten Lösung.

Grundsätzlich kann man sagen, dass der Grad der Plattformunabhängigkeit umso höher ist desto geringer der Aufwand ist, mit dem eine Lösung zwischen Plattformen gewechselt werden kann (siehe auch A.1). Darüber hinaus geht in die Bewertung natürlich auch ein, wie hoch der Grad der Umsetzbarkeit der IT-Gesamtarchitektur[6] und den damit verbundenen Architekturvorschriften[7] ist sowie, wie stark die Entscheidungen der Behörde aufgrund bestehender Plattformabhängigkeit eingeschränkt werden (siehe auch Einleitung zu Kapitel 2).

---

1 siehe www.kbst.bund.de , Wirtschaftlichkeit und Recht / Wirtschaftlichkeit / Dokumente

2 siehe WiBe 4.0, Version 4.0, August 2004, Band 68 der Schriftenreihe der KBSt, Abschnitte 3.1 und 4.3

3 siehe WiBe 4.0, Version 4.0, August 2004, Band 68 der Schriftenreihe der KBSt, Abschnitte 3.1 und 4.4

4 siehe WiBe 4.0, Version 4.0, August 2004, Band 68 der Schriftenreihe der KBSt, Abschnitte 3.1 und 4.5

5 Innerhalb der Wibe 4.0: Bereich: Qualitativ-strategische Kriterien / Kriteriengruppe: Priorität der IT-Maßnahme / Kriterium: 4.1.5 (siehe www.kbst.bund.de, WiBe 4.0,)

6 Die IT-Gesamtarchitektur im Leitfaden entspricht der Architektur in Tabelle 3. Sie beschreibt die Architektur einer Behörde über die Gesamtheit ihrer Anwendungen (siehe vorausgehende Kapitel).

7 Die hier beschriebenen Architekturvorschriften entsprechen den Architekturvorgaben in Tabelle 3. Sie umfassen die internen Vorgaben einer Behörde zur Umsetzung ihrer Architektur (siehe vorausgehende Kapitel)

Tabelle 3: Kriterium Plattform-/Herstellerunabhängigkeit[1]

0	2	4	6	8	10
Nicht von Bedeutung bzw. keine ersichtlichen Wirkungen zu erwarten	Geringfügige qualitative Verbesserungen ohne strategisches Gewicht (z.B. Lösung steht in mehreren Versionen für unterschiedliche Plattformen zur Verfügung (Pseudo-unabhängigkeit))	Software kann mit geringfügigem Aufwand auf andere Plattformen portiert werden. Vorhandene Hardware/ Peripherie kann auch weiterhin in den geplanten Fristen eingesetzt werden.	Plattform-/ Hersteller-unabhängigkeit ist gewährleistet und die angestrebte Lösung trägt zur Erweiterung der Aus- und Umbau-optionen bei.	Plattform- / Hersteller-unabhängigkeit und Investitionsschutz sind gewährleistet, Vorgaben aus der IT-Architektur werden eingehalten.	Weitgehende Gestaltungsautonomie verbunden mit der Weiternutzung vorhandener Hard- und Software.

Null Punkte werden dann vergeben, wenn die Fachanwendung nicht plattformunabhängig ist und insbesondere dann, wenn dadurch die Entscheidungsfreiheit bezüglich strategischer Zielsetzungen (z.B. Vorgaben aus der IT-Gesamtstrategie) deutlich eingeschränkt wird.

Mit zwei Punkte kann das Kriterium bewertet werden, wenn eine eingeschränkte Plattformunabhängigkeit (siehe auch Tabelle 1) oder eine Pseudounabhängigkeit vorliegt. Wenn eine Fachanwendung plattformabhängig ist, aber Versionen für unterschiedliche Plattformen zur Verfügung stehen, dann spricht man von Pseudo- oder Scheinunabhängigkeit.

Lässt sich die Fachanwendung mit geringfügigem Aufwand auf verschiedene Plattformen portieren, dann ist eine Bewertung mit 4 Punkten gerechtfertigt. Dies ist zum Beispiel dann gegeben, wenn das Design und das Konzept einer Anwendung auf unterschiedlichen Plattformen einheitlich eingesetzt werden kann und nur einzelne technische Details, wie z.B. die Zugriffe auf das jeweilige Dateisystem anzupassen sind.

Ist die Plattformunabhängigkeit einer Fachanwendung gegeben aber sie passt nicht in allen Punkten in die Vorgaben der IT-Gesamtstrategie, sei es, dass nicht alle geforderten Standards eingehalten werden, und sie schränkt die Umsetzung der strategischen Ziele der der IT-Gesamtstrategie nicht ein, dann kann man eine Bewertung mit 6 Punkten vornehmen.

[1] siehe hierzu auch Tabelle 1

Passt die Fachanwendung zusätzlich noch vollständig in das Konzept IT-Gesamtstrategie und die aufgestellten Architekturvorschriften werden alle erfüllt, dann sollte die Lösung mit 8 Punkten bewertet werden.

Wenn darüber hinaus noch der Idealzustand der völligen Gestaltungsautonomie erreicht wird, dann kann die volle Punktzahl vergeben werden. Dies bedeutet, dass z.B. die strategische Weiterentwicklung der IT-Gesamtarchitektur absolut unabhängig von der Einführung der Fachanwendung vorgenommen werden kann.

4.2.3 Rechtliche Aspekte

4.2.3.1 Einleitung

Bei der Beschaffung plattformunabhängiger Fachanwendung sind die Vorgaben des europäischen und deutschen Vergaberechts zu beachten. Ziel des Vergaberechts ist ein wirtschaftlicher Einkauf, der durch Wettbewerb sichergestellt werden soll. Bei der vergaberechtlichen Beurteilung sind zwei Bereiche zu unterscheiden:

Für die Vergabe von Aufträgen mit einem geschätzten Auftragswert oberhalb der in der Vergabeverordnung (VgV) niedergelegten so genannten EG-Schwellenwerte gelten die in §§ 97 ff. des Gesetzes gegen Wettbewerbsbeschränkungen (GWB) niedergelegten Grundsätze des Vergabeverfahrens. Zudem sind - je nach Auftraggeber - die Abschnitte 2 oder 3 oder 4 der Verdingungsordnung für Leistungen Teil A (VOL/A) oder bei freiberuflichen Leistungen die VOIF anzuwenden. Dieser so genannte oberschwellige Bereich des Vergaberechts ist mit einem effektiven zweistufigem Rechtschutzsystem ausgestaltet, §§ 102 ff. GWB. Ein übergangener Bieter kann vor Vergabekammern die Einhaltung der ihn schützenden Bestimmungen über das Vergabeverfahren durchsetzen und vergaberechtswidrige Zuschlagserteilungen verhindern.

Für die (zahlenmäßig häufigen) Auftragsvergaben unterhalb der Schwellenwerte gelten die Bestimmungen des GWB nicht. Maßgebend für die Vergabe in diesem Bereich sind Vorschriften des Haushaltsrechts des Bundes bzw. der Länder. Zum Teil sehen diese die Anwendung des ersten Abschnitts der VOL/A, also die so genannten Basisparagraphen, verbindlich vor, zum Teil wird dies nur empfohlen. Etliche Länder haben in jeweils eigenen Vergabegesetzen weitere bzw. ergänzende oder auch davon abweichende Regelungen getroffen. Im Übrigen sind ggf. noch innerdienstliche Anweisungen zu beachten. In diesem unterschwelligen Bereich des Vergaberechts gibt es nach bislang gefestigter Meinung praktisch

keinen effektiven Primärrechtsschutz[1]. Jedoch können Fach- oder Rechtsaufsichtsbehörden als Nachprüfungsstellen formlos angerufen werden. Auch sind Schadensersatzansprüche des rechtswidrig unterlegenen Teilnehmers möglich[2].

Die schwierigen Fragen des Rechtsschutzes sind für die hier behandelte Problematik aber nicht entscheidend, steht doch außer Frage, dass sich die öffentliche Verwaltung an Verwaltungsvorschriften zu halten hat, wenn diese für die Vergabe öffentlicher Aufträge Geltung beanspruchen können. Dies wird sich nach dem derzeitigen Stand der Dinge auch in der anstehenden Reform des Vergaberechts nicht.

Die wesentlichen Verfahrensgrundsätze, auf die das Gutachten in erster Linie eingeht, namentlich der Gleichbehandlungsgrundsatz, das Transparenzgebot und der Wirtschaftlichkeitsgrundsatz, gelten somit im Regelfall unabhängig von Schwellenwerten für alle Auftragsvergaben. Im Folgenden soll untersucht werden, welche Vorgaben sich hieraus für die Beschaffung plattformunabhängiger Fachanwendungen ergeben.

4.2.3.2 Gleichbehandlungsgrundsatz

Der Gleichbehandlungsgrundsatz ist eine fundamentale Vorschrift des Vergaberechts. Er besagt, dass Teilnehmer an einem Vergabeverfahren in allen Stufen des Verfahrens gleich zu behandeln sind, es sei denn, eine Benachteiligung ist auf Grund dieses Gesetzes ausdrücklich gestattet, § 97 Abs. 2 Gesetz GWB, § 2 Nr. 2 VOL/A. Dieser Grundsatz wirft unterschiedliche Fragen auf, je nach dem, ob die Ausschreibung technische Spezifizierungen im Hinblick auf die gewünschte Plattformunabhängigkeit vorsieht (z. B. Java, J2EE oder LAMP) oder nicht.

Plattformunabhängigkeit ohne technische Spezifizierung

Werden Teilnehmer diskriminiert, wenn eine Beschaffungsstelle eine plattformunabhängige Fachanwendung ohne weitere technische Spezifizierung ausschreibt, wenn also die Leistungsbeschreibung keine Angaben dazu enthält, auf welche Weise die Plattformunabhängigkeit erreicht werden soll?

Auf den ersten Blick mag dies fern liegen. Der Gleichbehandlungsgrundsatz untersagt jedoch nicht nur offene Diskriminierungen von Teilnehmern, sondern auch

[1] Vgl. die Nachweise bei *Ruthig*, NZBau 2005, 497 ff. In einer viel beachteten Entscheidung hat das OVG Koblenz jüngst jedoch in einem nicht rechtskräftig gewordenen Beschluss (die Parteien schlossen im Hauptsacheverfahren einen Vergleich), geurteilt, dass unter bestimmten Voraussetzungen in einem nicht vom GWB erfassten Bereich des Vergaberechts, der Rechtsweg zu den Verwaltungsgerichten gegeben ist, vgl. OVG Koblenz, Beschl. v. 25.05.2005, NZBau 2005, 411. Es bleibt abzuwarten, ob sich diese Rechtsansicht durchsetzt. Klärung könnte eine beim BVerfG anhängige Verfassungsbeschwerde bringen, mit der das Fehlen eines effektiven Rechtsschutzes für Streitigkeiten unterhalb der EG-Schwellenwerte als Verstoß gegen Art. 3 und 19 IV GG angegriffen wurde (Az. 1 BvR 1160/03).

[2] Vgl. nur BGH, Urt. v. 8. 9. 1998, NJW 1998, 3636.

versteckte Ungleichbehandlungen.[1] Eine solche verdeckte Diskriminierung kann gegeben sein, wenn die ausgeschriebene Leistung technisch in einer Weise beschrieben ist, die bestimmte Anbieter von vorneherein als Teilnehmer ausschließt. § 8 Nr. 3 Abs. 4 der Verdingungsordnung für Leistungen (VOL/A) bestimmt näher: „Die Beschreibung technischer Merkmale darf nicht die Wirkung haben, dass bestimmte Unternehmen oder Erzeugnisse bevorzugt oder ausgeschlossen werden, es sei denn, dass eine solche Beschreibung durch die zu vergebende Leistung gerechtfertigt ist." Technische Merkmale können also nur festgelegt werden, wenn sie nicht bewirken, dass potentielle Bewerber aus unsachlichen Gründen ferngehalten werden.[2] Für die Zulässigkeit der Angabe technischer Merkmale ist es weder erforderlich, dass die beschriebene Leistung die objektiv einzig vertretbare Möglichkeit darstellt, noch dass es sich um die technisch optimale Lösung handelt. Vielmehr genügt es für die Bejahung der objektiven Erforderlichkeit, wenn die geforderte Eigenschaft sachlich vertretbar ist.[3]

Legt man diesen Maßstab zugrunde, so erweisen sich Ausschreibungen plattformunabhängiger Fachanwendungen als zulässig. Zwar wird diese Anforderung Anbieter vor Probleme stellen, die ausschließlich Anwendungen für nur eine spezifische Plattform anbieten, sei es, dass sie auch diese Plattform liefern und auf diese Weise ihre Marktposition sichern wollen, sei es, dass sie aus anderen Gründen nur Anwendungen für eine Plattform liefern. Die Benachteiligung der betroffenen Anbieter ist jedoch aus zwei Gründen gerechtfertigt. Zum einen werden sie nicht vollständig von der Ausschreibung ausgeschlossen, weil es ihnen offen steht, entsprechende Leistungen anzubieten, indem sie ihre Angebote umstellen. Zum anderen sprechen sachliche Gründe für die Forderung nach Plattformunabhängigkeit: Entsprechende Fachanwendungen können auf unterschiedlichen Plattformen eingesetzt werden, verhindern dadurch Produktabhängigkeiten und eröffnen Behörden die Möglichkeit, bei künftigen Beschaffungen unterschiedliche Angebote wahrnehmen zu können. Dadurch können Folgekosten vermieden werden. Die Forderung nach Plattformunabhängigkeit ist deshalb sachlich vertretbar und führt grundsätzlich nicht zu einer willkürlichen Benachteiligung von Teilnehmern.

Eine pauschale Forderung nach Plattformunabhängigkeit ohne Erklärung, was man darunter versteht und wozu man sie braucht ist jedoch nicht zulässig.

---

1 Immenga/Mestmäcker-*Dreher*, GWB, 3. Aufl. (2001), § 97 Rz. 49 ff.

2 VÜA Brandenburg, 09.05.2996, WUW/E VergAL 48 - *Heizkraftwerk Cottbus.*

3 Für die parallele Frage im Bauvergaberecht: *Ingenstau/Korbion*, VOB: Verdingungsordnung für Bauleistungen, Teile A und B, 13. Aufl., 1996, A § 9 Nr. 5, Rz. 84

Plattformunabhängigkeit mit technischer Spezifizierung

Es kann für Behörden von Interesse sein, bei der Ausschreibung einer Fachanwendung nicht nur allgemeine Plattformunabhängigkeit zu fordern, sondern auch im Einzelnen vorzugeben, durch welche Technologie diese erreicht werden soll. Bei dieser Konstellation ergeben sich aus dem Gesichtspunkt der Gleichbehandlung zusätzliche vergaberechtliche Fragen. Der Grundsatz ist - im Hinblick auf die Forderung bestimmter Technologien - durch zwei Bestimmungen in § 8 Nr. 3 VOL/A näher konkretisiert.

Vorgabe bestimmter Verfahren

§ 8 Nr. 3 Abs. 3 VOL/A sieht vor: „Bestimmte Erzeugnisse oder Verfahren sowie Ursprungsorte und Bezugsquellen dürfen nur dann ausdrücklich vorgeschrieben werden, wenn dies durch die Art der zu vergebenden Leistung gerechtfertigt ist." Als bestimmtes Verfahren ist die Art und Weise der Herstellung der Leistung oder bestimmter Eigenschaften der Leistung anzusehen.[1] Sinn und Zweck dieser Ausnahmevorschrift ist es, eine Verengung oder sogar Ausschaltung des Wettbewerbs durch eine einseitige Orientierung der Vergabestelle zu verhindern und den Grundsatz der Chancengleichheit zu wahren.[2] Das Vorschreiben bestimmter Verfahren ist nur dann gerechtfertigt, wenn dies aus technischer oder wirtschaftlicher Sicht sachlich vertretbar ist.[3] Dies kann beispielsweise der Fall sein, wenn bei objektiver Betrachtungsweise nur eine Technologie geeignet ist, mit der sonstigen Hard- und Software der Behörde zusammen zu arbeiten[4] oder wenn sich die Mitarbeiter bestimmtes Know-how zu einem Verfahren angeeignet haben. Denkbar ist auch, dass technisch überhaupt nur ein Verfahren in Frage kommt. Sind dagegen in Wirklichkeit mehrere Verfahren in gleicher Weise für die Erfüllung der Aufgabe geeignet, so kann die Begrenzung auf ein bestimmtes Verfahren gegen das Vergaberecht verstoßen.[5]

Im Ergebnis ergibt sich daraus das folgende Bild: Nur wenn ausnahmsweise sachliche Gründe für die Vorgabe einer bestimmten Technologie bestehen, so darf die Behörde die Ausschreibung entsprechend eng formulieren. Die Beweis-

---

1 Für die parallele Frage im Bauvergaberecht: *Ingenstau/Korbion*, VOB: Verdingungsordnung für Bauleistungen, Teile A und B, 13. Aufl., 1996, A § 9 Nr. 5, Rz. 82.

2 Daub/Eberstein-*Zdzieblo*, Kommentar zur VOL/A: Verdingungsordnung für Leistungen – ausgenommen Bauleistungen, Vergaberecht – Rechtsschutz, 5. Aufl. 2000, Abschnitt 1, § 8, Rz. 66 noch zur alten VOL/A, die sich an dieser Stelle durch die Neufassung von 2002 aber nicht geändert hat.

3 Für die parallele Frage im Bauvergaberecht: *Ingenstau/Korbion*, VOB: Verdingungsordnung für Bauleistungen, Teile A und B, 13. Aufl., 1996, A § 9 Nr. 5, Rz. 83.

4 So auch das Merkblatt des BMWA und des BMI „Diskriminierungsfreie Leistungsbeschreibung bei IT-Ausschreibungen" in Bezug auf Aufträge zum Einkauf von Mikroprozessoren. Das Kriterium der Kompatibilität ist im Bereich der Bauleistungen ebenfalls anerkannt, vgl. *Franke/Grünhagen u.a.*, VOB: Kommentar, 2002, § 9 VOB, Rz. 120.

5 VÜA Bund, 23.04.1997, WuW/E Verg 86 - *Schleusentechnik*.

last für das Vorliegen entsprechender Gründe liegt im Fall eines Nachprüfungsverfahrens bei der Behörde. Fehlt es an Gründen für die Vorgabe einer bestimmten Technologie, so ist es die Aufgabe der Bewerber, die für die Ausführung der Leistung notwendigen Verfahren auszuwählen und vorzuschlagen. Die Behörde kann in diesem Fall die technischen und wirtschaftlichen Kriterien jedoch bei der Wertung der Angebote gem. § 25 VOL/A berücksichtigen.

Verwendung bestimmter Bezeichnungen

Ist die Vorgabe eines bestimmten Verfahrens nach diesen Grundsätzen zulässig, so dürfen gemäß § 8 Nr. 3 Abs. 5 „Bezeichnungen für bestimmte Erzeugnisse oder Verfahren (z.B. Markennamen)“ nur ausnahmsweise benutzt werden, wenn eine Beschreibung durch hinreichend genaue, allgemeinverständliche Angaben nicht möglich ist und die Bezeichnung im technischen Sprachgebrauch mit der fraglichen Leistung gleichgesetzt wird. In diesem Fall ist der Zusatz „oder gleichwertiger Art“ zu verwenden. Da dies nur in Ausnahmefällen zulässig ist, sind hohe Anforderungen zu stellen. Behörden sollten daher versuchen, die Leistungsbeschreibung möglichst neutral zu formulieren, auch wenn es einfacher erscheint, einen bekannten Markennamen oder eine bekannte Technologie zu verwenden, mit der möglicherweise bereits gute Erfahrungen gemacht wurden.[1]

Allerdings ist zu konzedieren, dass Technologien, die für die Umsetzung der Plattformunabhängigkeit von Fachanwendungen Verwendung finden, aufgrund ihrer hohen Komplexität oft nicht mit allgemeinverständlichen Angaben beschrieben werden können. Hier kann es unverzichtbar sein, für die Leistungsbeschreibung auf übliche Bezeichnungen, beispielsweise Java, J2EE, Mono etc. zurückzugreifen. Allerdings muss in diesem Fall unbedingt der Zusatz „oder gleichwertiger Art“ in der Leistungsbeschreibung aufgenommen werden. Es muss zudem verhindert werden, dass die Ausschreibung offensichtlich auf nur einen Anbieter zugeschnitten ist. Dies gilt auch, wenn die Technologie von mehreren Unternehmen genutzt und angewandt wird, gleichwohl aber Lizenzgebühren für die Nutzung an ein Unternehmen geleistet werden müssen. Um Risiken zu vermeiden, sollte bei der Verwendung von entsprechenden Bezeichnungen stets der Zusatz aufgenommen werden.

4.2.3.3 Transparenzgebot

Mit dem Gleichbehandlungsgrundsatz eng verbunden ist der in § 97 Abs. 1 GWB verankerte Transparenzgrundsatz, der vorschreibt, dass alle Bieter zum Zeitpunkt der Erstellung ihrer Angebote über die gleichen Chancen verfügen. Auch soll durch eine möglichst umfangreiche Information der Bieter eine nachvollziehbare Gestaltung des Vergabeverfahrens dieser Grundsatz verwirklicht werden.

[1] Vgl. allg. das Merkblatt des BMWA und des BMI „Diskriminierungsfreie Leistungsbeschreibung bei IT-Ausschreibungen“ in Bezug auf Aufträge zum Einkauf von Mikroprozessoren.

Das Transparenzgebot hat sich in einer Reihe von konkreten Verpflichtungen im Vergabeverfahren niedergeschlagen:

Erstens hat die Vergabestelle darauf zu achten, dass in der Ausschreibung die Leistungsmerkmale im Einzelnen beschrieben sind, vgl. § 8 Nr. 1 Abs. 2 VOL/A. Nach der Rechtsprechung des Bundesgerichtshofs dürfen die Zuschlagskriterien nach Abgabe der Angebote nicht mehr verändert werden.[1] Bei der Beschaffung plattformunabhängige Fachanwendungen ist deswegen darauf zu achten, dass alle gewünschten technischen Merkmale oder Verfahren bereits in der Ausschreibung genau genug bestimmt werden. Kriterien, die in der Ausschreibung nicht genannt wurden, dürfen später bei der Entscheidung nicht berücksichtigt werden.

Zweitens trifft die Vergabestelle die Verpflichtung, das Vergabeverfahren zu dokumentieren, vgl. §110 Abs. 2 S. 1 GWB und § 30 VOL/A. Die Vergabestelle hat die einzelnen Stufen des Vergabeverfahrens einschließlich der Begründung der einzelnen Entscheidungen in einem Vergabevermerk zu dokumentieren.[2] Diese Dokumentation dient der Überprüfbarkeit der Vergabe durch die Rechnungsprüfungsbehörden und durch andere Bieter. Der Vergabestelle können die Unterlagen im Streitfall zudem als Beweis dafür dienen, dass die entsprechenden Sachüberlegungen bereits vor der Ausschreibung angestellt wurden. Bei der Ausschreibung plattformunabhängiger Fachanwendungen ist deswegen darauf zu achten, dass die Gründe für die Vorgabe bestimmter technischer Merkmale oder bestimmter Verfahren in den Vergabeakten dokumentiert werden.[3]

4.2.3.4 Wirtschaftlichkeitsgrundsatz

Bei der Wertung von Angeboten ist insbesondere der Wirtschaftlichkeitsgrundsatz gemäß § 97 Abs. 5 GWB bzw. § 25 VOL/A zu beachten. Diese Bestimmung geht von einem vierstufigen Wertungsverfahren aus. Danach ist zunächst zu prüfen, ob die Angebote wegen formaler Mängel, fehlender Bietereignung (fehlender Fachkunde, Leistungsfähigkeit und Zuverlässigkeit) oder unangemessen hoher oder niedriger Preise ausgeschlossen werden müssen. Unter den verbliebenen Angeboten erhält schließlich dasjenige den Zuschlag, das am wirtschaftlichsten ist. Gemäß § 25 Nr. 3 VOL/A muss der Zuschlag an „das unter Berücksichtigung aller Umstände günstige Angebot“ ergehen, wobei „der niedrigste Angebotspreis allein“ nicht entscheidend ist. Maßgeblich ist vielmehr das beste Preis-Leistungs-Verhältnis.[4] Das Angebot ist in diesem Sinne am wirtschaftlichsten, das die güns-

1 BGH, 08.09.1998, WuW/E Verg 150 - *Klärwerkerweiterung*.

2 Brandenburgisches OLG, 03.08.1999, WuW/E Verg. 238 - *Flughafen Berlin*.

3 Kapellmann/Messerschmidt-*Kapellmann/Langen*, § 9 VOB/A, 2003, Rz. 54.

4 Immenga/Mestmäcker-*Dreher*, GWB, 3. Aufl. (2001), § 97 Rz. 145.

tige Relation zwischen dem verfolgten Zweck und dem einzusetzenden Mittel aufweist.[1]

Bei der Bewertung der Eignung der Angebote im Rahmen der Wirtschaftlichkeitsprüfung ist strikt darauf zu achten, dass nur auftragsbezogene Kriterien herangezogen werden. Vergabefremde Kriterien, die nichts über die Leistungsfähigkeit der Angebote aussagen, etwa die allgemeine Intention, den Wettbewerb im Softwaremarkt zu stärken, Vorlieben für bestimmte Produkte oder Verfahren etc., sind unzulässig und dürfen in die Bewertung der Angebote nicht einfließen. Es ist stets zu fragen, ob sich das fragliche Bewertungskriterium auf den konkreten Auftrag bezieht oder nicht.[2]

Die dem deutschen Recht zugrunde liegende „Richtlinie 92/50/EWG des Rates vom 18.06.1992 über die Koordinierung der Verfahren zur Vergabe öffentlicher Dienstleistungsaufträge“[3] gibt in Art. 36 Abs. 1 lit. a) Kriterien an, anhand derer die Leistungsfähigkeit eines Angebots beurteilt werden kann: „Qualität, technischer Wert, Ästhetik, Zweckmäßigkeit der Leistung, Kundendienst und technische Hilfe, Lieferzeitpunkt, Ausführungszeitraum- oder frist, Preis.“ Der Europäische Gerichtshof hat zudem die zuverlässige und dauerhafte Versorgung als Zuschlagskriterium anerkannt.[4] Diese Kriterien dürfen zur Auslegung des deutschen Rechts, insbesondere § 97 Abs. 5 GWB herangezogen werden. Die genannten Kriterien decken sich im Wesentlichen auch mit den in § 25 VOL/A niedergelegten Kriterien und sind daher auch allgemein von Interesse, weil ihnen entnommen werden kann, dass nicht nur die Leistungsfähigkeit zum Zeitpunkt der Lieferung zu berücksichtigen ist, sondern auch Folgekosten („Kundendienst und technische Hilfe“).[5] So ist es beispielsweise allgemein anerkannt, dass höhere Wartungskosten bei der Bewertung der Angebote mit zu berücksichtigen sind.[6]

Eine in diesem Zusammenhang interessante Frage ist, ob die Vergabestelle einem Bieter, der für alle Teillose einer Leistung jeweils das günstige Angebot abgibt, den Zuschlag für einen Teil der Lose verweigern kann, um eine Abhängigkeit von diesem Unternehmen zu vermeiden. Das OLG Düsseldorf hat dies in der

---

1 Begründung des Regierungsentwurfs Vergaberechtsänderungsgesetz, Bundestags-Drucksache 13/9340, S. 14.

2 Immenga/Mestmäcker-*Dreher*, GWB, 3. Aufl. (2001), § 97 Rz. 146.

3 Amtsblatt EG Nr. L 209/1 vom 24.07.1992, S. 1.

4 EuGH, 28.03.1995, Rs. C-324/93, Slg. 1995 I, 610, Rz. 44 - *Queen/Home Department*.

5 Vgl. Daub/Eberstein-*Kulartz,* Kommentar zur VOL/A: Verdingungsordnung für Leistungen – ausgenommen Bauleistungen, Vergaberecht – Rechtsschutz, 5. Aufl. 2000, Abschnitt 1, § 25, Rz 37. Für die parallele Frage im Bauvergaberecht auch Kapellmann/Messerschmidt-*Dähne*, § 25 VOB/A, 2003, Rz. 69 f.; *Franke/Grünhagen u.a.*, VOB: Kommentar, 2002, § 25 VOB, Rz. 58.

6 Kapellmann/Messerschmidt-*Dähne* a.a.O. (zur Vergabe von Bauleistung); Daub/Eberstein-*Kulartz,* a.a.O. (allg. zur VOL/A).

Entscheidung „Euro-Münzplättchen III“ bejaht.[1] In der Literatur ist diese Entscheidung zu Recht kritisiert worden, da die Unabhängigkeit bei künftigen Anschaffungen nichts über die Wirtschaftlichkeit der aktuellen Angebote aussagt.[2] Es ist - jenseits der gesetzlichen Ausnahmen, beispielsweise § 97 Abs. 3 GWB (Mittelstandsförderung) - nicht die Aufgabe der einzelnen Vergabestellen, die Strukturen der Beschaffungsmärkte zu manipulieren.

Legt man diesen Maßstab an, so erweist sich die Unabhängigkeit einer Fachanwendung von einer bestimmten Plattform als zulässiges Bewertungskriterium. Die technische Unabhängigkeit ist eine Eigenschaft der Leistung, die Folgekosten vermeidet. Abhängigkeiten bei künftigen Beschaffungen werden auf diese Weise vermieden. Anders als in der Entscheidung „Euro-Münzplättchen III“[3] wird diese Unabhängigkeit nicht dadurch erreicht, dass ein an sich ebenso leistungsfähiges und günstigeres Angebot benachteiligt wird. Die technische Unabhängigkeit ist hier vielmehr eine Eigenschaft des Angebots selbst. Wäre die Berücksichtigung der entsprechenden Eigenschaft unzulässig, so wäre es beispielsweise nicht möglich, Busse mit Hybridantrieb (Diesel und Elektromotor) gegenüber herkömmlichen mit Dieselmotoren ausgestatteten Bussen zu bevorzugen, da auch in diesem Fall künftige Kosten über die Leistungsfähigkeit der aktuellen Anschaffung entscheiden. Sofern die wahrscheinliche Vermeidung künftiger Kosten unmittelbare Folge der Eigenschaften eines Angebots ist, so ist diese Eigenschaft als Kriterium zulässig. Die Plattformunabhängigkeit kann bei der Bewertung der Angebote also mitberücksichtigt werden.

1 OLG Düsseldorf, 15.06.2000, NZBau 2000, 440 - *Euro-Münzplättchen III*.

2 Immenga/Mestmäcker-*Dreher*, GWB, 3. Aufl. (2001), § 97 Rz. 160.

3 Siehe Immenga/Mestmäcker-*Dreher*, GWB, 3. Aufl. (2001), § 97 Rz. 160..

5 Zusammenfassung und Tools für die praktische Umsetzung

Wie die vorangehenden Kapitel gezeigt haben, ist Plattformunabhängigkeit ein Begriff mit zahlreichen Facetten. Vor dem Kauf oder der Entwicklung einer plattformunabhängigen Fachanwendung bedarf es daher einer genauen Spezifizierung, was unter Plattformunabhängigkeit im Detail verstanden wird und welchen Zielen die Plattformunabhängigkeit dient. Gewisse Abhängigkeiten einer Fachanwendung, beispielsweise von einer Technologieplattform, sind im Lichte anderer gewünschter Eigenschaften sowie von Randbedingungen, die sich aus der strategischen Ausrichtung der Behörde ergeben, häufig unvermeidbar. Diese Abhängigkeiten können jedoch durch geeignete Maßnahmen reduziert und gesteuert werden. Plattformunabhängigkeit ist insofern keine Entscheidung zwischen Gut und Böse, sondern eine herausfordernde Gestaltungsaufgabe.

In **Tabelle 4** werden verschiedene Hinweise zusammengefasst, die in den vorangehenden Kapiteln gegeben wurden, jeweils mit einem Verweis auf diejenige Stelle im Dokument, die den adressierten Aspekt von Plattformunabhängigkeit genauer behandelt.

Tabelle 5 listet Standards auf, die im Zusammenhang mit Plattformunabhängigkeit von Bedeutung sind.

Tabelle 6 listet rechtliche Empfehlungen auf.

5.1 Zusammenfassende Tabellen

Tabelle 4: Maßnahmen zur Durchsetzung von Plattformunabhängigkeit

Maßnahmen zur Durchsetzung von Plattformunabhängigkeit			
Kontext	**Ziel**	**Maßnahme**	**Fundstellen**
Definition einer Anwendungsarchitektur	Sicherstellung der isolierten Lauffähigkeit der Anwendung	Realisierung der Anwendung als Einzelplatzanwendung (siehe auch Praxistool 3 und Praxistool 4)	Abschnitt 3.2.2
	Zentrale Speicherung und Validierung von Daten	Realisierung der Anwendung als Client/Server-Anwendung (siehe auch Praxistool 3 und Praxistool 4)	Abschnitt 3.2.2
	Bereitstellung verschiedener Anwendungen unter einer gemeinsamen Oberfläche (Portal)	Realisierung der Anwendungen als Client/Server-Anwendungen (siehe auch Praxistool 3 und Praxistool 4)	Abschnitt 3.2.2
	Sicherstellung der Lauffähigkeit einer Client/Server-Anwendung bei nicht-kontinuierlicher Kommunikationsverbindung	Implementierung des Clients als Fat-Client (siehe auch Praxistool 3 und Praxistool 4)	Abschnitt 3.2.2
	Vereinfachung der Software-Verteilung bei einer Anwendung, die vielen Nutzern zur Verfügung gestellt werden soll oder häufig aktualisiert werden muss	Implementierung des Clients als Thin-Client (siehe auch Praxistool 3 und Praxistool 4)	Abschnitt 3.2.2

Maßnahmen zur Durchsetzung von Plattformunabhängigkeit			
Kontext	**Ziel**	**Maßnahme**	**Fundstellen**
Definition einer Anwendungsarchitektur	Reduzierung der Plattformabhängigkeit	Verwendung offener, allgemein anerkannter Standards (siehe auch Tabelle 5)	Abschnitt 3.2.4
Definition einer Systemarchitektur	Feststellung, ob fachliche Abhängigkeiten zwischen den Anwendungen der IT-Landschaft bestehen (siehe auch Praxistool 1 und Praxistool 2 sowie Praxistool 7 und Praxistool 8)	Identifikation von Fachanwendungen, die Arbeitsergebnisse anderer Fachanwendungen benötigen (siehe auch Praxistool 7)	Abschnitt 4.1.3.1,
		Identifikation von Abhängigkeiten zwischen den Daten verschiedener Anwendungen	Abschnitt 4.1.3.1
		Identifikation fachlich gleichwertiger Funktionen, die von mehreren Anwendungen ausgeführt werden	Abschnitt 4.1.3.1
	Anwendungen ohne jegliche Abhängigkeiten untereinander (siehe auch Praxistool 8)	Realisierung der Fachanwendungen als Monolithen in Ausnahmefällen möglich. Besser auf Basis von n-Schichten-Architektur und komponentenbasiert. (siehe auch Praxistool 5)	Abschnitt 4.1.3.1
	Vermeidung von Integration (z.B. aus rechtlichen Gründen (z.B. Datenschutz))	Realisierung der Fachanwendungen als Monolithen in Ausnahmefällen möglich. Besser auf Basis von n-Schichten-Architektur und komponentenbasiert. (siehe auch Praxistool 5)	Abschnitt 4.1.3.1

Maßnahmen zur Durchsetzung von Plattformunabhängigkeit			
Kontext	**Ziel**	**Maßnahme**	**Fundstellen**
Definition einer Systemarchitektur	Integration kleiner IT-Landschaften mit wenig Abhängigkeiten	Verbinden der Anwendungen durch Punkt-zu-Punkt-Integrationen. (siehe auch Praxistool 5 und Praxistool 16)	Abschnitt 4.1.3.2
	Reduzierung der Abhängigkeit von Integrationssoftware (DOT-Middleware, MOM, ESB) (siehe auch Praxistool 9 bis Praxistool 20)	Verwendung von Produkten, die allgemein anerkannten Standards folgen (siehe auch Tabelle 5)	Abschnitt 4.1.3.3
		Flankierende organisatorische Maßnahmen	Abschnitt 4.1.3.3,
		Berücksichtigung der Frage, wie hoch der Lernaufwand für die Administration bei der Beschaffung einer Integrationssoftware ist	Abschnitt 4.1.3.3
		Beachtung der üblichen Regeln bezüglich der Plattformunabhängigkeit von Betriebssystem und Datenbank-Management-System	Abschnitt 4.1.3.3 Abschnitt 3.2.2
	Integration von Anwendungen, die mit Hilfe unterschiedlicher Technologiefamilien implementiert wurden	Aufbau einer serviceorientierten Architektur (SOA) (siehe auch Praxistool 9Praxistool 12)	Abschnitt 4.1.3.4

Maßnahmen zur Durchsetzung von Plattformunabhängigkeit			
Kontext	**Ziel**	**Maßnahme**	**Fundstellen**
	Integration von Anwendungen, deren weitere Entwicklung nicht oder nur in geringem Maße beeinflusst werden kann	Aufbau einer serviceorientierten Architektur (SOA) (siehe auch Praxistool 9Praxistool 12)	Abschnitt 4.1.3.4
Definition einer Systemarchitektur	Integration von Anwendungen, die Dienste anderer Organisationen nutzen oder Dienste für andere Organisationen bereit stellen	Aufbau einer serviceorientierten Architektur (SOA) (siehe auch Praxistool 9Praxistool 12)	Abschnitt 4.1.3.4
	Reduzierung der Aufrufe eines Services aus Performanzgründen	Definition grobgranularer Schnittstellen und kontextfreier Funktionalitäten für Services (siehe auch Praxistool 11Praxistool 12)	Abschnitt 4.1.3.4
	Orchestrierung von Services für die Umsetzung änderungsanfälliger Prozesse	Einsatz eines Enterprise Service Bus (ESB) (siehe auch Praxistool 19)	Abschnitt 4.1.3.4
	Orchestrierung von Services mit langen oder nicht vorhersagbaren Antwortzeiten	Einsatz einer Message-oriented Middleware (MOM, ESB) (siehe auch Praxistool 19 und Praxistool 20)	Abschnitt 4.1.3.4
	Orchestrierung von Services, deren permanente Verfügbarkeit nicht garantiert ist	Einsatz einer Message-oriented Middleware (MOM, ESB) (siehe auch Praxistool 19 und Praxistool 20)	Abschnitt 4.1.3.4

Maßnahmen zur Durchsetzung von Plattformunabhängigkeit			
Kontext	**Ziel**	**Maßnahme**	**Fundstellen**
	Orchestrierung von Services, deren Aufenthaltsort sich gelegentlich ändern kann	Einsatz einer Message-oriented Middleware (MOM, ESB) (siehe auch Praxistool 19 und Praxistool 20)	Abschnitt 4.1.3.4
	Orchestrierung von Services für die Umsetzung von Verfahren, bei denen verschiedene beteiligte Komponenten parallel arbeiten können	Einsatz einer Message-oriented Middleware (MOM, ESB) (siehe auch Praxistool 19 und Praxistool 20)	Abschnitt 4.1.3.4
Definition einer Systemarchitektur	Orchestrierung von Services für die Umsetzung von Verfahren, bei denen Ereignisse auftreten können, über die eine im vorhinein nicht bekannte Menge von Services informiert werden muss	Einsatz einer Message-oriented Middleware (MOM, ESB) (siehe auch Praxistool 19 und Praxistool 20)	Abschnitt 4.1.3.4
	Orchestrierung von Services für die Umsetzung von Verfahren, bei denen Altanwendungen integriert werden müssen	Einsatz eines Enterprise Service Bus (ESB) und Zuhilfenahme von Adaptern	Abschnitt 4.1.3.4
	Realisierung einer Einheit (Service oder Fachanwendung), die entweder gar nicht mit anderen Anwendungen und Komponenten interagieren muss oder nur mit solchen, die mit Hilfe der gleichen Technologiefamilie implementiert werden kann	Definition einer komponentenbasierten Architektur	Abschnitt 4.1.3.5

Maßnahmen zur Durchsetzung von Plattformunabhängigkeit			
Kontext	**Ziel**	**Maßnahme**	**Fundstellen**
	Realisierung eines Services, für den jetzt oder zukünftig Komponenten benötigt werden, bei denen kein Einfluss auf die verwendete Implementierungstechnologie genommen werden kann	Zerlegung des Services in Teilservices Serviceorientierte Integration und komponentenbasierte Realisierung der Teilservices	Abschnitt 4.1.3.5
	Umsetzung technischer Aspekte wie Transaktions-, Sicherheits- oder Persistenzmanagement im Rahmen einer komponentenbasierten Architektur	Einsatz einer Distributed Object Middleware (DOT-Middleware) (siehe auch Praxistool 20)	Abschnitt 4.1.3.5

Maßnahmen zur Durchsetzung von Plattformunabhängigkeit			
Kontext	**Ziel**	**Maßnahme**	**Fundstellen**
Aufbau und Nutzung von technischem Know-how	Harmonisierung der durch skriptende Power-User vorgenommenen Softwareentwicklung mit der Strategischen Landschaftsarchitektur	Zusammenführen der skriptenden Power-User in einer Gruppe	Abschnitt 4.1.4.2
		Schulung der Mitglieder dieser Gruppe in den Grundkonzepten der Softwareentwicklung	Abschnitt 4.1.4.2
		Beauftragung der Gruppe mit bestimmten Aktivitäten, wie z.B. der Inventarisierung der vorhanden Skripte, Makros und Templates oder der Mitwirkung an der weiteren Entwicklung der Strategischen Landschaftsarchitektur	Abschnitt 4.1.4.2
Entwicklung (Eigen- oder Auftragsentwicklung) einer Fachanwendung	Sicherstellung der erwünschten Plattformunabhängigkeit	Auswahl einer geeigneten Programmierumgebung (Programmiersprache, Entwicklungswerkzeuge)	Abschnitt 3.1.4,
		Untersuchung von Programmiersprachen daraufhin, ob die Ergebnis-Programme für ihre Ausführung eine Black-Box benötigen und ob eine solche Black-Box für verschiedene Plattformen verfügbar ist.	Abschnitt 3.1.4

Maßnahmen zur Durchsetzung von Plattformunabhängigkeit			
Kontext	**Ziel**	**Maßnahme**	**Fundstellen**
Entwicklung (Eigen- oder Auftragsentwicklung) einer Fachanwendung	Sicherstellung der erwünschten Plattformunabhängigkeit	Aufnahme der für die Reduzierung von Plattformabhängigkeiten zuständigen Anforderungen in Spezifikation und Pflichtenheft und als Kriterium in der Wirtschaftlichkeitsbetrachtung	Abschnitt 4.2.1.1 Abschnitt 4.2.2
		Präzise Definition, bezüglich welcher Plattformen und in jeweils welchem Maße Unabhängigkeit gewünscht wird	Abschnitt 4.2.1.1
		Erleichterung der Verifikation des Ergebnisses durch Einflussnahme auf den Entwicklungsprozess	Abschnitt 4.2.1.2,
	Reduzierung der Abhängigkeit von einer Technologieplattform	Etablierung eines Modell-getriebenen Vorgehens	Abschnitt 4.2.1.2
Kauf einer Fachanwendung	Sicherstellung der erwünschten Plattformunabhängigkeit	Durchführung einer Marktanalyse unter Verwendung der spezifizierten Anforderungen als Evaluationskriterien	Abschnitt 4.2.1.3
		Aufnahme der für die Reduzierung von Plattformabhängigkeiten zuständigen Anforderungen in Spezifikation und Ausschreibungsunterlage	Abschnitt 4.2.1.1

Maßnahmen zur Durchsetzung von Plattformunabhängigkeit			
Kontext	**Ziel**	**Maßnahme**	**Fundstellen**
Kauf einer Fachanwendung	Sicherstellung der erwünschten Plattformunabhängigkeit	Präzise Definition, bezüglich welcher Plattformen und in jeweils welchem Maße Unabhängigkeit gewünscht wird	Abschnitt 4.2.1.1
		Berücksichtigung der Anforderungen bei Bewertung der Angebote und der Zuschlagsentscheidung	Abschnitt 4.2.1.4

Tabelle 5: Standards und Technologien

Standards				
Kontext	**Standard**	**Aufgabe**	**Kategorisierung in SAGA 3.0**	**Fundstellen**
Serviceorientierte Architekturen	SOAP	Protokoll für die Kommunikation zwischen Service-Nehmer und Service-Anbieter	obligatorisch (Version 1.1)	Abschnitt 3.2.5 Abschnitt 4.1.3.4
	WSDL	Formale Beschreibung der Schnittstellen eines Web-Services	obligatorisch (Version 1.1)	Abschnitt 4.1.3.4
	XML	Format für den Austausch von Daten	obligatorisch (Version 1.0)	Abschnitt 4.1.3.4
	XSD	Spezifikation zu übertragender Datenelemente	obligatorisch (Version 1.0)	Abschnitt 4.1.3.4
	OSCI-Transport	Querschnittsaufgaben im Sicherheitsbereich bei der Abwicklung von Transaktionen mittels Web-Services	obligatorisch (Version 1.2)	Abschnitt 4.1.3.3 Abschnitt 4.1.3.4
	XKMS	Registrierung und Verteilung öffentlicher Schlüssel zum Zwecke sicherer, XML-basierter Kommunikation	empfohlen (Version 2)	Abschnitt 4.1.3.4
	WS-Security	Sicherung der Vertraulichkeit, Integrität und Verbindlichkeit von SOAP-Nachrichten	empfohlen (Version 1.1)	Abschnitt 4.1.3.4
	UDDI	Verzeichnisdienste für Web-Services	unter Beobachtung (Version 2.0)	Abschnitt 4.1.3.4
	BPEL4WS	Prozessbeschreibungssprache für die Orchestrierung von Web-Services	unter Beobachtung (Version 1.1)	Abschnitt 4.1.3.3 Abschnitt 4.1.3.4

Standards				
Kontext	**Standard**	**Aufgabe**	**Kategorisierung in SAGA 3.0**	**Fundstellen**
Serviceorientierte Architekturen	WS-CDL	XML-basierte Schnittstellenbeschreibungs-sprache für die Orchestrierung von Web-Services	Whitelist	Abschnitt 4.1.3.4
	SAML	Austausch von Authentifizierungs- und Autori-sierungsinformationen bei der Kommunikation zwischen Web-Services	empfohlen (Version 2.0)	Abschnitt 4.1.3.4
Komponentenbasierte Architekturen	J2EE	Familie von Komponententechnologien für den Einsatz in Enterprise-Anwendungen	obligatorisch (Version 1.4)	Abschnitt 4.1.3.4 Abschnitt 4.1.3.5
	J2SE	Familie von Komponententechnologien für den Einsatz in allen Arten von Anwendungen	obligatorisch (Version 1.4)	Abschnitt 4.1.3.4 Abschnitt 4.1.3.5
	XML	Format für den Austausch von Daten	obligatorisch (Version 1.0)	Abschnitt 4.1.3.2
Komponentenbasierte Architekturen	.NET	Komponentenframework	unter Beobachtung (Version 2.0)	Abschnitt 4.1.3.4 Abschnitt 4.1.3.5
Entwicklung und Pflege einer IT-Gesamtarchitektur	UML	Notationssprache für Modelle und Architekturen	empfohlen (Version 2.0)	Abschnitt 4.1.4.2

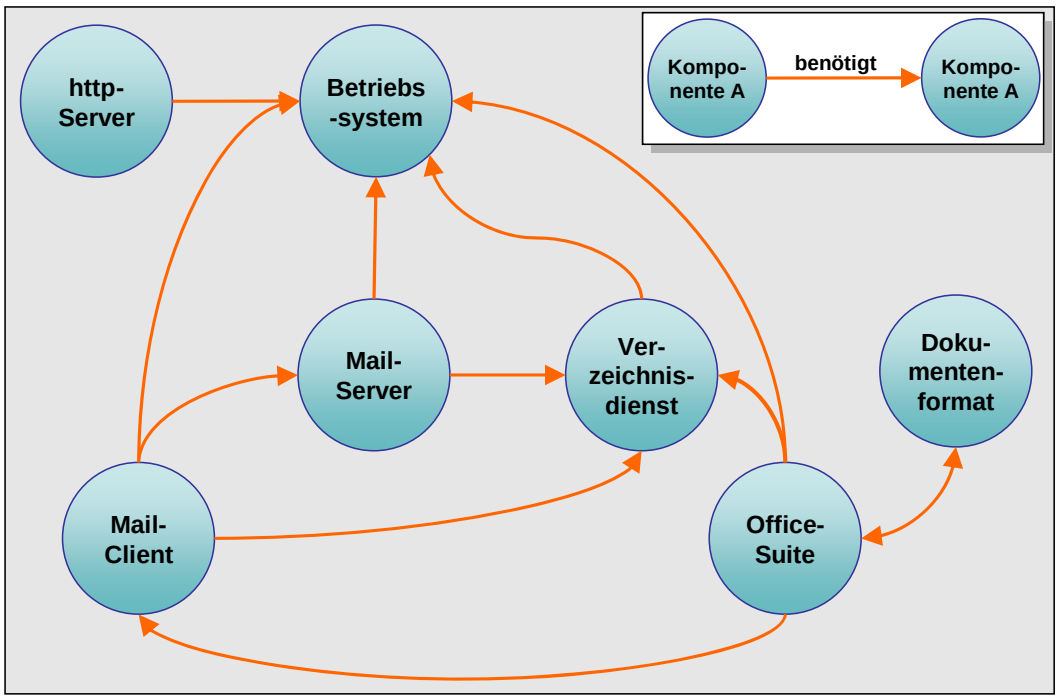
Tabelle 6: Rechtliche Empfehlungen

Rechtliche Empfehlungen	
Empfehlung	**Fundstelle**
Die Ausschreibung plattformunabhängiger Fachanwendungen ohne technische Spezifizierung ist vergaberechtlich zulässig. Technische Spezifizierungen dürfen nur gefordert werden, wenn hierfür ein sachlicher Grund besteht.	Abschnitt 4.2.1.1 Abschnitt 4.2.3.2 Seite 120
Die Nennung bestimmter Bezeichnungen, insbesondere Markennamen ist nur ausnahmsweise zulässig, wenn die Leistung auf andere Weise nicht beschrieben werden kann. In diesem Fall ist der Zusatz „oder gleichwertiger Art“ aufzunehmen.	Abschnitt 4.2.3.2, Seite 123
Die gewünschten technischen Merkmale oder Verfahren müssen bereits in der Ausschreibung genau genug bestimmt werden. Kriterien, die in der Ausschreibung nicht genannt wurden, dürfen später bei der Entscheidung nicht berücksichtigt werden.	Abschnitt 4.2.1.1, Seite 99 Abschnitt 4.2.3.3, Seite 123
Bei der Ausschreibung plattformunabhängiger Fachanwendungen ist darauf zu achten, dass die Gründe für die Vorgabe bestimmter technischer Merkmale oder bestimmter Verfahren in den Vergabeakten dokumentiert werden	Abschnitt 4.2.3.3, Seite 124
Die Plattformunabhängigkeit von Fachanwendungen kann bei der Bewertung der Angebote mitberücksichtigt werden.	Abschnitt 4.2.1.1, Seite 99 Abschnitt 4.2.3.4, Seite 127

5.2 Praxistools

5.2.1 Identifizieren von Abhängigkeiten und Unabhängigkeiten

Praxistool 1: Abhängigkeitsgraph (siehe Kapitel 2 und vor allem 2.1.6)



Praxistool 2: Frageliste zur Identifizierung von Abhängigkeiten und deren Ausprägung (siehe Kapitel 3.1und 3.2)

Frageliste

Aufgabe: Festellen der Ausprägung von Plattformunabhängigkeit einer Fachanwendung.

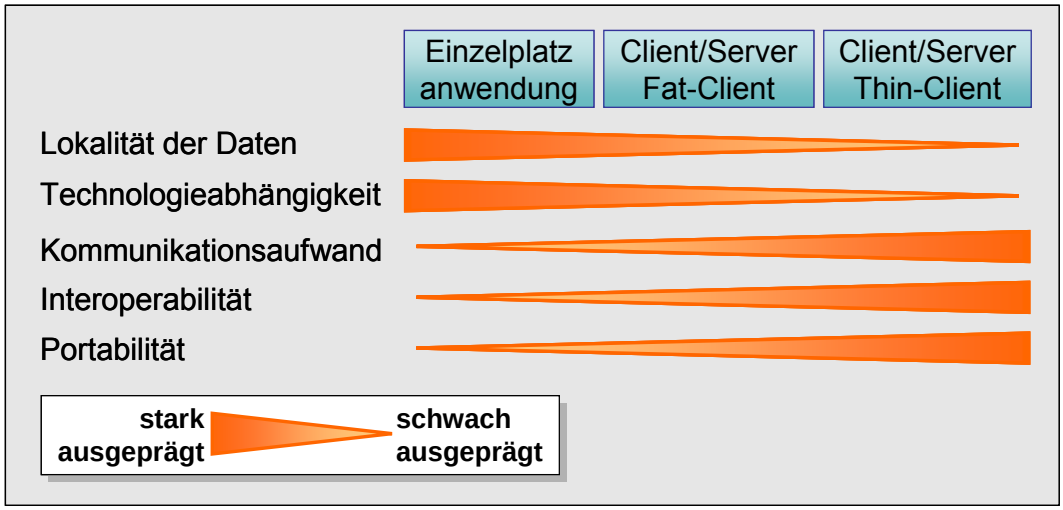
1. Verwendet die Fachanwendung ein Entwicklungs- und Laufzeitsystem zur Entkopplung vom Basis-System?
2. Ist das durch die Fachanwendung verwendete Laufzeitsystem auf mehreren Basis-Systemen lauffähig?
3. Kann die Fachanwendung unabhängig von einer bestimmten Version der Laufzeitumgebung genutzt werden?
4. Folgen die Schnittstellen zur Anbindung der Fachanwendung an andere Systeme anerkannten Standards?
5. Sind die Schnittstellen öffentlich dokumentiert?
6. Sind die entsprechenden APIs offen gelegt und dokumentiert?

7. Ist das Laufzeitsystem weitgehend unabhängig vom Basis-System?
    a. Wird bestimmte Hardware benötigt bzw. darf bestimmte Hardware nicht eingesetzt werden?
    b. Wird Infrastruktursoftware, beispielsweise Datenbank-Management-Software, benötigt?
    c. Müssen spezielle Produkte eines bestimmten Herstellers genutzt werden?
    d. Muss diese Software in bestimmten Versionen vorliegen?
8. Falls die Fachanwendung an andere Anwendungen gebunden ist, ist die Bindung (Nutzung) so gestaltet, dass die Fachanwendung auch ohne diese anderen Anwendungen sinnvoll genutzt werden kann?

Immer wenn die Hauptfragen (1 bis 9) mit nein beantwortet werden, besteht eine mehr oder weniger hohe Gefährdung der Plattformunabhängigkeit. Je mehr der Hauptfragen mit nein beantwortet werden, umso geringer ist die Ausprägung der Plattformunabhängigkeit. Handelt es sich um eine bestehende Fachanwendung, dann sollte kurz- bis mittelfristig Abhilfe geschaffen werden. Entweder an den betroffenen Stellen (Antwort = nein) oder durch Austausch der Fachanwendung. Worauf hierbei zu achten ist, wird in den vorangegangen und nachfolgenden Kapiteln beschrieben.

5.2.2 Auswahl von Anwendungstyp, Architekturmuster, Programmiersprache und Schnittstellen

Praxistool 3: Übersichtsgrafik zur Auswahl des Anwendungstypen (siehe Kapitel 3.2.2)



Praxistool 4: Grundsätze zur Auswahl des richtigen Anwendungstyps (siehe Kapitel 3.2.2)

1. Anwendungen, die isoliert lauffähig sein müssen, sollten als Einzelplatzanwendungen realisiert werden.
2. Client/Server-Anwendungen eignen sich optimal für Anwendungen, bei denen Daten zentral gespeichert und validiert werden.
3. Client/Server-Anwendungen, die sich nicht auf eine kontinuierliche Kommunikationsverbindung zwischen Client und Server verlassen können, sollten einen Fat-Client nutzen.
4. Haben Client/Server-Anwendung sehr viele Benutzer, die darüber hinaus im Wesentlichen Aufgaben der reinen Datenerfassung bzw. des Datenabrufs durchführen, eignet sich ein Thin-Client besonders gut.
5. Werden Fachanwendungen aus fachlichen Gründen häufig aktualisiert, lassen sich diese im Normalfall bei einer Client/Server-Anwendung, speziell bei Verwendung eines Thin-Clients, besonders gut durchführen.
6. Werden verschiedene einzelne Anwendungen unter einer gemeinsamen Oberfläche verbunden, eignet sich der Client/Server Ansatz gut. Meist wird die gemeinsame Oberfläche dabei als Thin-Client realisiert. Häufig spricht man in diesem Zusammenhang auch von „Portalen“.

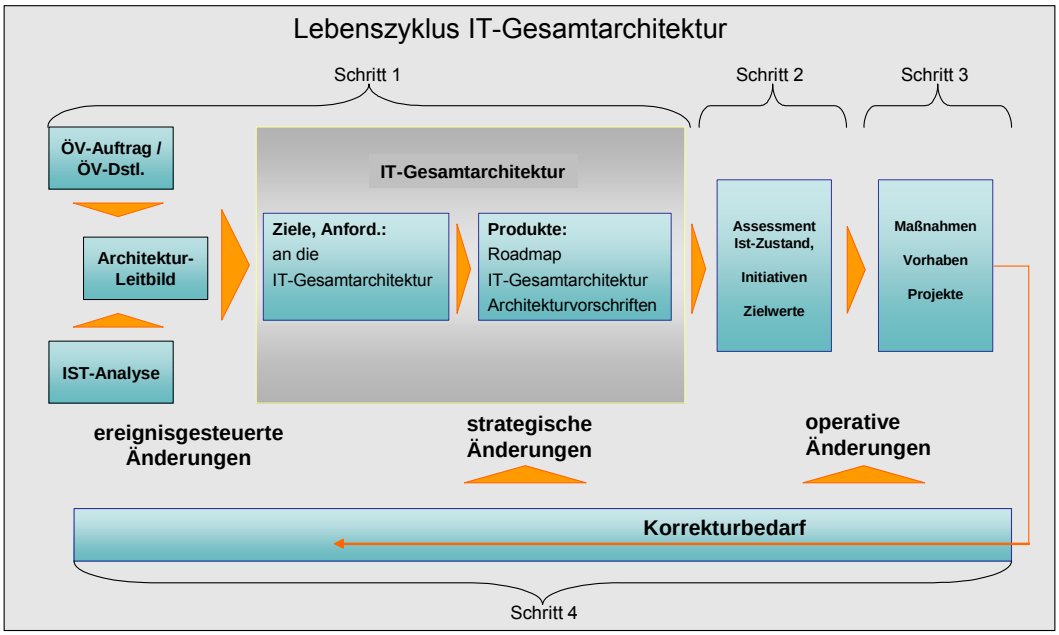
Praxistool 5: Empfehlungen zu Anwendungstyp-, Architekturmuster-, Programmiersprachen- und Schnittstellenwahl (siehe Kapitel 3.2.6)

Thema	Empfehlungen
Anwendungstyp	Die Wahl für eine Einzelplatz- oder eine C/S-Anwendung ist stark abhängig von den jeweiligen fachlichen Anforderungen. Wenn möglich sollte der Typ einer Client/Server-Anwendung und hier vor allem die Webanwendung oder das Portal gewählt werden.
Architekturmuster	Gut geeignet sind n-Schichten- und Service orientierte Architekturen. Komponentenbasierte Architekturen können ebenfalls genutzt werden, besitzen aber die Einschränkung, dass i.d.R. die Bindung zwischen den Komponenten höher ist. Monolithen sind mit Blick auf die Plattformunabhängigkeit eher weniger geeignet.
Programmiersprache	Hier ist Java zu empfehlen, da Java für viele Betriebssysteme verfügbar ist. Dies gilt nicht für .NET, das in einer vollständigen Implementierung nur für Windows verfügbar ist.

Thema	Empfehlungen
Schnittstellen	Schnittstellen sind immer dann gut geeignet, wenn sie offenen Standards folgen. Die Erfahrungen haben gezeigt, dass insbesondere XML-basierte Schnittstellen und Protokolle sich gut eigenen, um Unabhängigkeit zu erreichen.

5.2.3 Gestaltung der IT-Gesamtarchitektur

Praxistool 6: Übersicht strategischer Prozess zur Gestaltung der IT-Gesamtarchitektur (siehe Kapitel 4.1.1 und 4.1.2)



5.2.4 Wahl der IT-Gesamtarchitektur

(siehe 4.1.3.1 bis 4.1.3)

Praxistool 7: Fragen zur fachlichen Abhängigkeit

- Gibt es oder wird es in Zukunft fachliche Abhängigkeiten zwischen den Anwendungen geben?
- Gibt es oder wird es in Zukunft Fachanwendungen geben, die Arbeitsergebnisse anderer Fachanwendungen benötigen?
- Gibt es oder wird es in Zukunft Abhängigkeiten zwischen den Daten verschiedener Anwendungen geben?
- Gibt es oder wird es in Zukunft fachlich gleichwertige Funktionen geben, die von mehreren Anwendungen ausgeführt werden?

Praxistool 8: Regel zum Umgang mit fehlenden fachlichen Abhängigkeiten

Auch wenn nach gründlicher Prüfung feststeht, dass keinerlei fachliche Abhängigkeiten bestehen, ist eine Silo-Architektur dennoch aus den in Abschnitt 3.2.3.1 genannten Gründen nach Möglichkeit zu vermeiden..

Praxistool 9: Fragen zur Integrationsnotwendigkeit

- Müssen jetzt oder in Zukunft heterogene Anwendungen und Komponenten integriert werden, d.h. Anwendungen und Komponenten, die mit Hilfe unterschiedlicher Technologiefamilien (z.B. Java und .NET) implementiert sind?
- Wie stark kann die weitere Entwicklung der zu integrierenden Anwendungen und Komponenten beeinflusst werden?
- Gibt es Schnittstellen nach außen, d.h., werden Dienste anderer Organisationen genutzt oder Dienste für andere Organisationen bereit gestellt, so dass eine Integration mit diesen sinnvoll ist?

Praxistool 10: Regel zur Auswahl einer SOA

Müssen Anwendungen und Komponenten integriert werden, die nicht in einer einheitlichen Technologie implementiert wurden oder für die dies nicht auf Dauer garantiert werden kann, bietet sich eine serviceorientierte Architektur an.

Praxistool 11: Hinweise zur Definition von Service-Schnittstellen

Die wichtigste Aufgabe beim Entwurf einer serviceorientierten Architektur ist die Definition geeigneter Schnittstellen. Hier ist folgendes zu beachten:

- Die Kommunikation mit einem Service ist vergleichsweise langsam. Deshalb sollte aus Performanzgründen die Anzahl der benötigten Aufrufe nicht zu groß werden.
  - Services werden über eine Netzwerkverbindung angesprochen. Daher ist die Kommunikation mit einem Service mit einer gewissen Latenzzeit verbunden.
  - Für die mittels Services realisierten Behördenprozesse gelten häufig hohe Sicherheitsanforderungen, u.a. weil häufig mit personenbezoge-

nen Daten gearbeitet wird. Es müssen deshalb Sicherheitsmechanismen (Datenverschlüsselung, Authentifizierung, Autorisierung) in die Kommunikation eingebaut werden.

- XML als universelles Datenformat gestattet es, beliebige Nutzdaten in strukturierter Form zu versenden. Allerdings verwenden die beteiligten Anwendungen intern häufig ein anwendungsspezifisches, kompakteres Format. Für den Datenaustausch müssen die Daten in diesen Fällen umgewandelt werden.

❍ HTTP und SOAP sind so genannte zustandslose Protokolle, d.h. nach einer Anfrage und einer Antwort ist der Dialog zwischen Client und Service prinzipiell beendet. Ein zustandsbehaftetes Protokoll in diesem Sinne würde weitere Anfrage-/Antwort-Paare vorsehen, von denen jedes inhaltlich auf die Ergebnisse der vorausgegangenen aufbauen könnte, also auf den bereits erreichten Zustand der Verbindung. Eine zustandsorientierte Kommunikation auf der Basis zustandsloser Protokolle ist selbstverständlich möglich; da die Verwaltung des Zustandes jedoch nicht durch das Protokoll geschieht, muss sie durch die Anwendungslogik geleistet werden, und trägt so zu deren Komplexität bei.

Praxistool 12: Regel für die Definition von Service-Schnittstellen

Services sollten grobgranulare Schnittstellen und kontextfreie Funktionalitäten anbieten.

Praxistool 13: Regel für die Auswahl von komponentenbasierten Architekturen

Für die Integration im Kleinen sind serviceorientierte Architekturen jedoch weniger geeignet. Hier eigenen sich komponentenbasierte Architekturen besser

Praxistool 14: Regel gegen die komponentenbasierte Realisierung eines Service

Wenn die Möglichkeit besteht, dass für die Umsetzung des Services jetzt oder zukünftig Komponenten benötigt werden, bei denen kein Einfluss auf die verwendete Implementierungstechnologie genommen werden kann, dann sollte der Service nicht komponentenbasiert umgesetzt werden.

Praxistool 15: Hinweise zur Definition von Services und Komponenten

- Services sollten mit Standard-Technologien umgesetzt werden. Das betrifft das Kommunikationsprotokoll (SOAP), das Transportprotokoll (HTTP), die verwendeten Datenformate (XML) und die Spezifikation der Schnittstellen (WSDL, XSD).
- Die bei der Spezifikation von Service-Schnittstellen verwendeten XML-Schemata sollten an einer zentralen Stelle koordiniert werden.
- Bei der Spezifikation der Service-Schnittstellen sollte auf die richtige Granularität der Schnittstellen geachtet werden. Jeder Service muss eine kontextfreie Funktionalität anbieten.
- Für die Implementierung einzelner Services bieten sich komponentenbasierte Architekturen an.[1]
- Als Komponententechnologie ist Java (J2SE/J2EE) gemäß SAGA obligatorisch, während .NET den Status "unter Beobachtung" hat[2].
- Besteht die Möglichkeit, dass für die Umsetzung des Services jetzt oder zukünftig Komponenten benötigt werden, bei denen kein Einfluss auf die verwendete Implementierungstechnologie genommen werden kann, sollte der Service in geeignete Teilservices zerlegt werden.

1 Das gilt auch dann, wenn es sich um die Umsetzung einer vollständigen Fachanwendung handelt, die entweder gar nicht mit anderen Anwendungen und Komponenten interagieren muss oder nur mit solchen, die zur IT-Landschaft derselben Behörde gehören und mit Hilfe der gleichen Technologiefamilie implementiert wurden.

2 Zur Bedeutung der Klassifizierung dieser Technologien siehe Kapitel „Klassifizierung und Lebenszyklen von Standards“ in SAGA 3.0 (www.kbst.bund.de)

Praxistool 16: Regel zum Einsatz von Integrationssoftware

Enthält die IT-Landschaft nur eine Handvoll Fachanwendungen, können diese durch Punkt-zu-Punkt-Integrationen (P2P-Integrationen, siehe Abschnitt 3.2.3) ohne Verwendung einer zentralen Integrationssoftware miteinander verbunden werden.
Zeichnet sich bereits ab, dass eine derzeit noch kleine IT-Landschaft in absehbarer Zeit um weitere Anwendungen wachsen wird, sollte auf P2P-Integrationen verzichtet werden.
Kommen P2P-Integrationen aus den genannten Gründen nicht in Betracht, sollte eine zentrale Integrationssoftware zur Koordinierung der Integrationen eingesetzt werden.

Praxistool 17: Hinweise für die Beschaffung von Integrationssoftware

- Die verwendete Integrationssoftware sollte allgemein anerkannten Standards folgen.
- Sie sollte so wenig wie möglich an proprietären Funktionen beinhalten.
- Die Integrationssoftware sollte weder von einem bestimmten Betriebssystem noch von einem bestimmten Datenbank-Management-System abhängig sein.
- Die Administration der Integrationssoftware sollte möglichst leicht zu erlernen sein.

Praxistool 18: Fragen zur Wahl für oder gegen den Einsatz eines ESB

- Werden änderungsanfällige Prozesse umgesetzt?
- Müssen Services genutzt werden, die lange oder nicht vorhersagbare Antwortzeiten haben?
- Sind Anwendungen oder Komponenten beteiligt, deren permanente Verfügbarkeit nicht garantiert ist?
- Werden Verfahren abgebildet, bei denen verschiedene beteiligte Komponenten parallel arbeiten können?
- Wird Flexibilität hinsichtlich der Lokalisation von Services benötigt?
- Gibt es Ereignisse, bei deren Auftreten eine unbekannte Zahl von Services existiert, die über das Ereignis informiert werden müssen?
- Müssen Legacy-Anwendungen (Altanwendungen) integriert werden?

Praxistool 19: Hinweise zum Einsatz eines ESB

- Wird mindestens eine der folgenden Fragen mit "Ja" beantwortet, sollte der Einsatz eines ESB[1] in Erwägung gezogen werden:
  - Werden änderungsanfällige Prozesse umgesetzt?
  - Müssen Services genutzt werden, die lange oder nicht vorhersagbare Antwortzeiten haben?
  - Sind Anwendungen oder Komponenten beteiligt, deren permanente Verfügbarkeit nicht garantiert ist?
  - Werden Verfahren abgebildet, bei denen verschiedene beteiligte Komponenten parallel arbeiten können?
  - Wird Flexibilität hinsichtlich der Lokalisation von Services benötigt?
  - Gibt es Ereignisse, bei deren Auftreten eine unbekannte Zahl von Services existiert, die über das Ereignis informiert werden müssen?
  - Müssen Legacy-Anwendungen (Altanwendungen) integriert werden?
- Wird ein ESB eingesetzt, sollte das ausgewählte Produkt den allgemeinen Anforderungen an eine Integrationssoftware genügen (siehe Abschnitt 4.1.3.3).

Praxistool 20: Hinweise zum Einsatz von DOT-Middleware

- Die verwendete DOT-Middleware sollte den allgemeinen Anforderungen an Integrationssoftware genügen (siehe Abschnitt 4.1.3.3).
- Bei der Beschaffung oder Entwicklung zukünftiger Komponenten für die Umsetzung des Services ist darauf zu achten, dass diese mit Hilfe derjenigen Technologie entwickelt werden, die der DOT-Middleware zugrunde liegt, und nicht auf ein bestimmtes DOT-Middleware-Produkt angewiesen sind, weil sie beispielsweise proprietäre Funktionalitäten dieses Produkts verwenden.

5.2.5 Weitere Praxistools

Weitere Praxistools, wie z.B. zum Erstellen einer Wirtschaftlichkeitsbetrachtung oder der Erstellung eines Kriterienkataloges nach UfAB werden auf den Web-Seiten der KBSt (www.kbst.bund.de) bereitgestellt.

[1] siehe „Enterprise Service Bus Seite 81

Anhang

A.1 WiBe 4.0 Kriterium - „Plattform-/Herstellerunabhängigkeit“

4.4.1.5 Plattform-/Herstellerunabhängigkeit

Mit diesem Kriterium bewerten Sie, inwieweit die angestrebte Lösung es erlaubt, (auch) künftig einerseits auf unterschiedlichen Plattformen eingesetzt werden zu können und andererseits weitere Ausbaustufen der IT-Architektur[1] möglichst frei und ohne Vorgaben des Hard- oder Softwareherstellers bzw. bestehender oder zukünftig geplanter Plattformen gestalten und auf verschiedene Anbieter zurückgreifen zu können.

Plattformunabhängige Lösungen stellen sich mit Blick auf die Einführung aus monetärer Sicht häufig weniger günstig dar als vergleichbare Lösungen, die auf proprietäre Plattformen festgelegt sind. Plattformunabhängigkeit bezieht sich dabei auf unterschiedliche Typen von Plattformen:

- Hardware
- Betriebssystem
- Infrastruktursoftware (z.B. Datenbank Management System)
- Standardsoftware (z.B. Officeanwendungen)
- Entwicklungsplattformen.

Plattformunabhängigkeit verfolgt eher einen (mittel- bis) langfristigen strategischen Ansatz. Mit plattformunabhängigen Lösungen kann sowohl der Produktlebenszyklus als auch die Einsatzdauer verlängert werden. Damit überwiegen dann die wirtschaftlichen Vorteile einer plattformunabhängigen Lösung in der Zukunft, wenn einerseits eine Überarbeitung oder gar der Austausch der Lösung nicht mehr nur durch den Austausch einer Plattform notwendig wird und andererseits die Plattform bei Bedarf (z.B. aus wirtschaftlichen Gründen) ausgewechselt werden kann, ohne dass gleichzeitig alle darauf aufsetzenden Lösungen ausgewechselt werden müssen.

Umso geringer der Aufwand ist, mit dem eine Lösung zwischen Plattformen gewechselt werden kann, desto höher der Grad der Plattformunabhängigkeit und i.d.R. auch höher der Grad der Herstellerunabhängigkeit (sofern es unterschiedliche Anbieter von Plattformen gibt).

[1] Architektur einer Behörde über die Gesamtheit ihrer Anwendungen.

4.4.1.5 Plattform-/Herstellerunabhängigkeit

0	2	4	6	8	10
Nicht von Bedeutung bzw. keine ersichtlichen Wirkungen zu erwarten	Geringfügige qualitative Verbesserungen ohne strategisches Gewicht (z.B. Lösung steht in mehreren Versionen für unterschiedliche Plattformen zur Verfügung (Pseudounabhängigkeit))	Software kann mit geringfügigem Aufwand auf andere Plattformen portiert werden. Vorhandene Hardware/ Peripherie kann auch weiterhin in den geplanten Fristen eingesetzt werden.	Plattform-/ Herstellerunabhängigkeit ist gewährleistet und die angestrebte Lösung trägt zur Erweiterung der Aus- und Umbauoptionen bei.	Plattform-/ Herstellerunabhängigkeit und Investitionsschutz sind gewährleistet, Vorgaben aus der IT-Architektur[1] werden eingehalten.	Weitgehende Gestaltungsautonomie verbunden mit der Weiternutzung vorhandener Hard- und Software.

[1] Interne Vorgaben einer Behörde zur Umsetzung ihrer Architektur, Festlegung von Standards, Technologien, Schnittstellen usw.

A.2 Abkürzungsverzeichnis

API	Application Programming Interface
APC	Arbeitsplatzcomputer
ASP	Application Service Provider
B2B	Business to Business
B2C	Business to Consumer
B2G	Business to Government
BPEL	Business Process Execution Language
BSD	Berkeley System Distribution
BSI	Bundesamt für Sicherheit in der Informationstechnik
BVA	Bundesverwaltungsamt
C2C	Consumer to Consumer
C2G	Consumer 2 Government
CORBA	Common Object Request Broker Architecture
COTS	Commercial off the Shelf
DBMS	Database Management System
DOMEA	Dokumentenmanagement und elektronische Archivierung
EfA	Einer für Alle / Einige für Alle
EIA	Enterprise Application Integration
EICTA	European Information and Communications Technology Industry Association
FMS	File Management System
FTP	File Transfer Protocol
G2B	Government to Business
G2C	Government to Citizens
G2G	Government to Government
GPL	GNU General Public License
HTTP	Hyper Text Transfer Protocol
IDABC	Interoperable Delivery of European eGovernment Services to Public Administrations, Businesses and Citizens
IT	Information und Telekommunikation
J2EE	Java 2 Plattform, Enterprise Edition

Abkürzungsverzeichnis

JDBC	Java Database Connectivity
LAMP	Linux, Apache, MySQL, PHP
MIT	Massachusetts Institute of Technology
MOM	Message Oriented Middleware
OASIS	Organization for Advancement of Structured Information Standards
ODF	Open Document Format
OOM	Object Oriented Middleware
OSCI	Online Services Computer Interface
OSI	Open Source Initiative
OSS	Open Source Software
PC	Personal Computer
PDF	Portable Document Format
RMI	Remote Message Invocation
SAGA	Standards und Architekturen für E-Government-Anwendungen
SMTP	Simple Mail Transfer Protocol
SOA	Service Oriented Architecture
SOAP	Simple Object Access Protocol
SQL	Structured Query Language
SW	Software
TCP/IP	Transmission Control Protocol/Internet Protocol
UML	Unified Modelling Language
VPS	Virtuelle Poststelle
WS	Web Services
WSDL	Web Services Description Language
XML	Extensible Markup Language
XÖV	XML-Schemata für die Öffentlichen Verwaltung
XSD	XML Schema Definition
XSL	Extensible Stylesheet Language
XSLT	XSL Transformations
ZVP	Zahlungsverkehrsplattform

A.3 Glossar

API

Application Programming Interface (Definierte Programmierschnittstelle, die für die Integration und Erweiterung genutzt werden kann)

Basiskomponenten

Eine Basiskomponente ist eine Komponente, die organisationsweit und hochverfügbar zur Verfügung stehen muss und einen Dienst bereitstellt, der von einer Vielzahl von Komponenten/Prozessen/Abläufen benötigt wird

BPEL (Business Process Execution Language)

BPEL ist eine auf XML-basierte Sprache für die Beschreibung und Ausführung von Workflows. Dabei können beliebig viele verteilte Anwendungen und Dienste im Internet, die über eine Web-Service-Schnittstelle verfügen, in einen Workflow integriert werden. Die ursprüngliche Version von BPEL ist eine gemeinsame Entwicklung von Microsoft, IBM, SAP AG, BEA Systems und Siebel Systems, die zur Standardisierung eingereicht wurde und gegenwärtig von OASIS (Organization for the Advancement of Structured Information Standards) weitergeführt wird. In BPEL werden neben der „Workflow"-Beschreibung eines Geschäftsprozesses auch alle für den Ablauf notwendigen Zugriffe und Schnittstellen definiert.

Browser

Browser sind zunächst einmal Computerprogramme, mit denen man Textdateien betrachten kann. Ein spezieller und viel verwendeter Browser ist der Webbrowser. Hierbei handelt es sich um ein Computerprogramm mit dem man Web-Seiten, z.B. im Internet, betrachten und zwischen diesen navigieren kann. Der Web-Browser interpretiert dabei so genannte HTML-Dokumente und stellt diese auf dem Bildschirm dar.

C#

C# (gesprochen „ci sharp") ist eine objektorientierte Programmiersprache, die vom Hersteller Microsoft entwickelt wurde. Sie weist neben Konzepten und Strukturen aus Java und Delphi auch Konzepte aus den Programmiersprachen C++ und Visual Basic auf. C# ist die favorisierte Programmiersprache für .NET.

Dienst (Service)

Unter dem Begriff des Dienstes selbst ist eine Entität zu verstehen, die eine abgeschlossene Aufgabe erfüllt und ihre Funktionalität über eine Dienstschnittstelle bereitstellt. Im Gegensatz zu Komponenten wird von den Abhängigkeiten der die Dienste erbringenden Entitäten abstrahiert. So kann es sich bei der Entität um eine Komponente handeln, die von einem Diensteanbieter entwickelt, gewartet und angeboten wird, es kann sich aber auch um eine Legacy-Anwendung han-

deln, die (Teile) ihre(r) Funktionalität über eine Dienstschnittstelle anbietet. Dienste gliedern sich oft nahtlos in eine Middleware ein, die dafür sorgt, dass die Dienstnutzer einen Dienst finden und mit ihm interagieren können.

DOMEA-Konzept

Das DOMEA-Konzept ist das Konzept für Dokumenten-Management und elektronische Archivierung in der öffentlichen Verwaltung. Es wird verantwortet durch die KBSt. Wesentliches Ziel des DOMEA-Konzeptes ist die Einführung der elektronischen Akte. An die Stelle der Papierakten sollen künftig behördliche Geschäftsprozesse treten, die medienbruchfrei und vollständig elektronisch realisiert werden können. Die elektronische Akte wird in IT-gestützter Vorgangsbearbeitung erzeugt, erfasst und verwaltet. Dabei gelten die gleichen Anforderungen an das elektronische Schriftgut, die in Gesetzen, Geschäftsordnungen, sowie Richtlinien und Vorschriften für die Papierakten festgelegt sind. Behördliche Unterlagen müssen auch in elektronischer Form den Kriterien Vollständigkeit, Integrität und Authentizität, Zusammenfassung aufgabenbezogener und zusammengehöriger Schriftstücke, Nachvollziehbarkeit und Rechtmäßigkeit des Verwaltungshandelns genügen. So müssen auch elektronische Akten hinreichenden Inhalt und Struktur aufweisen und sich in einen Kontext einordnen lassen. Elektronische Akten sollen wie ihre Vorgänger im Papierformat über die unmittelbare Bearbeitung hinaus ihre Nachweisfunktion erfüllen.

eGovernment

Als allgemein anerkannte Definition von eGovernment gilt die Definition der Deutschen Hochschule für Verwaltungswissenschaften Speyer: „eGovernment ist die Abwicklung geschäftlicher Prozesse im Zusammenhang mit Regieren und Verwalten (Government) mit Hilfe von Informations- und Kommunikationstechniken über elektronische Medien."

eBusiness

eBusiness wird allgemein als Oberbegriff für elektronische Geschäftsprozesse im eCommerce und im eGovernment benutzt. eBusiness beschreibt die Bereiche B2B (Business to Business), B2C (Business to Consumer), C2C (Consumer to Consumer), B2G (Business to Government), C2G (Consumer to Government) und G2G (Government to Government). Obwohl der Begriff eBusiness diesbezüglich nicht explizit eingeschränkt wird, wird er meist im Zusammenhang mit der Nutzung des Internet als Kommunikationsmedium verwendet.

EfA-Dienstleistung

Einer-für-Alle-Dienstleistungen, siehe www.kbst.bund.de/saga-efa)

Erweiterbarkeit

Erweiterbarkeit in der Softwaretechnik beschreibt den Grad der Möglichkeit Änderungen an Software einfach, zielorientiert und wirtschaftlich durchführen zu

können. Die Komplexität einer Software ist der maßgebliche Faktor, der die Erweiterbarkeit der Software beeinflusst.

Fachanwendung

Ein Software-System, das Mitarbeiter bei der Umsetzung von Dienstleistungen oder internen Prozessen der Verwaltung unterstützt.

Hersteller- / (Dienst-)anbieterunabhängigkeit

Der Grad der Flexibilität, den Anbieter bezüglich alternativer Produkte und Dienste jederzeit wechseln zu können.

Interoperabilität

Das Thema Interoperabilität beschäftigt sich mit den organisatorischen und technischen Grundlagen der engen Verzahnung von IT-Systemen. Dementsprechend lautet die Definition der IDABC-EIF[1]: „Interoperabilität beschreibt die Möglichkeiten von Informations- und Kommunikationssystemen, Daten, Informationen und Wissen miteinander auszutauschen..."

Diese sehr allgemein gehaltene Definition lässt sich dadurch spezifizieren, dass verschiedene Dimensionen von Interoperabilität identifiziert werden:

- Organisatorische Interoperabilität
- Semantische Interoperabilität
- Technische Interoperabilität
- Dokumenteninteroperabilität

Die **Organisatorische Interoperabilität** beschreibt den rein organisatorischen und rechtlichen – nicht technischen – Abstimmungsprozess zwischen den verschiedenen Partnern, im Fall von eGovernment z. B. den Prozess zwischen Behörden, die ihre Daten austauschen wollen. Dieser Abstimmungsprozess beinhaltet beispielsweise die Definition gemeinsamer Zielstellungen, die Analyse der vorhandenen Geschäftsprozesse, die Planung und übergreifende Modellierung der neu zu schaffenden Geschäftsprozesse und die Durchführung von Bedarfsanalysen zu Themen wie Sicherheit, Verfügbarkeit, Abrechenbarkeit und Nutzerfreundlichkeit.

Die **Semantische Interoperabilität** beschäftigt sich mit der präzisen Definition der auszutauschenden Informationen.

Dabei gilt, dass die dafür zu spezifizierenden Datensätze möglichst allgemeingültig und gut verständlich aufgebaut sein sollten. Das Ziel dabei ist, dass sie zu einem späteren Zeitpunkt auch von anderen ursprünglich nicht eingeplanten Sys-

[1] Definition gem. European Interoperability Framework for Pan-European eGovernment Service, Version 1.0, November 2004

temen nutzbar sind. Semantische Interoperabilität bzw. die Eindeutigkeit der Informationen spielt nicht nur bei der „Maschine zu Maschine“-Kommunikation eine entscheidende Rolle. Auch bei mehrsprachiger Umsetzung von Inhalten, zum Beispiel im europäischen Kontext, ist ein gemeinsames und eindeutiges semantisches Verständnis jedes einzelnen Elementes einer Information von Bedeutung.

Im Gegensatz zu den bislang vorgestellten, eher organisatorischen Dimensionen umfasst die **technische Interoperabilität** alle Problemstellungen rund um das Verbinden von verschiedenen technischen Systemen, Anwendungen oder „Diensten“. Entsprechend beschäftigen sich die Themengebiete dieser Dimension beispielsweise mit offenen Schnittstellen, Verbindungsdiensten, Übertragungswegen und -protokollen, Integration von verschiedenen Datenformaten, Software-Technologien und Middleware-Plattformen und Themen wie Authentifizierung und sichere Datenübertragung. Letztlich werden hier alle Bereiche angesprochen, die benötigt werden, um die Komponenten einer IT-Infrastruktur tatsächlich zu integrieren.

Darüber hinaus beschäftigt sich die **Dokumenteninteroperabilität** zusätzlich mit dem Thema des Austauschs von Informationen aus Office-Anwendungen.

Im Allgemeinen wird unter Dokumenteninteroperabilität die Eigenschaft von Desktop-Anwendungen verstanden, die in den jeweiligen Dokument-Typen enthaltenen Daten extrahieren und in vorgegebene, möglichst standardisierte XML-Strukturen umwandeln zu können. Damit ist Dokumenteninteroperabilität zur Erreichung von Plattformunabhängigkeit besonders für Standardanwendungen wie beispielsweise Office-Systeme, besonders wichtig.

Der Grundgedanke dabei ist, dass die so gewonnenen XML-Datensätze zwischen den verschiedenen Systemen ausgetauscht und weiterverarbeitet werden können. Die Dokumenteninteroperabilität ist dabei aber jedoch nicht nur auf den Austausch von Daten zwischen Office-Systemen oder anderen Desktop-Anwendungen beschränkt.

Letztlich werden durch die Dokumenteninteroperabilität beispielsweise Office-Anwendungen in die Lage versetzt, direkt mit verschiedenen eGovernment-Diensten, -Plattformen und Fachanwendungen zu kommunizieren. Damit ist sie die Grundlage für viele aktuelle Aufgabenstellungen wie das Steuern von Business-Prozessen aus Office-Anwendungen oder die Einbindung von Office-Anwendungen in entsprechende eGovernment-Prozesse.

Investitionssicherheit
Nachhaltigkeit des Mitteleinsatzes

Java

Java ist eine objektorientierte, plattformunabhängige Programmiersprache und wurde von der Firma Sun Microsystems entwickelt.

Durch die Einführung einer „Ausführungsumgebung“, der Java Virtual Machine, die für nahezu alle Computer-Betriebssysteme erhältlich ist, sind Java-

Programme auf nahezu allen Computern und auch auf verschiedenen anderen technischen Geräten (z. B. PDA oder Mobiltelefon) lauffähig.

Java 2 Enterprise Edition

Java 2 Enterprise Edition (J2EE) ist ein Oberbegriff für verschiedene Konzepte und Java-basierte Komponenten, die insbesondere bei dem Betrieb von J2EE-Application-Servern genutzt werden.

Neben der Client-seitigen Kommunikation mit J2EE-Application-Servern, die meist mit einem Browser erfolgt, unterstützt J2EE auch eine Kommunikation zwischen Anwendungskomponenten (Applications). Für die anwendungsseitige Kommunikation können verschiedene Techniken eingesetzt werden: XML-basierte Web-Services, CORBA und direkte Aufrufe aus Java-Programmen.

Java Server Pages

Java Server Pages sind ein Teil der J2EE-Technologie. Mit ihnen wird insbesondere die Entwicklung der Präsentationsschicht (dynamische Internetseiten) durchgeführt. Java Server Pages sind im wesentlichen Textdokumente, die statischen Text (in Form von HTML) und dynamische Anteile (JSP-Elemente) beinhalten. Die dynamischen Anteile werden während der Laufzeit von einem Application-Server ausgewertet und durch entsprechenden Text oder Zahlen ersetzt.

Java Servlets

Java Servlets sind ein Teil der J2EE-Technologie. Mit ihnen wird (wie auch mit den Java Server Pages) insbesondere die Entwicklung der Präsentationsschicht (von dynamischen Internetseiten) durchgeführt.

Java Virtual Machine

Die Java Virtual Machine ist die Laufzeitumgebung für Java Programme.

JDBC

Java Database Connectivity ist die einheitliche Java-Anwendungsschnittstelle für Datenbanken verschiedener Hersteller. Über diese Schnittstelle können SQL-Befehle ausgeführt und mit SQL-basierten Datenbanken kommuniziert werden. Über JDBC kann mit Java direkt auf ODBC-Datenbanken zugegriffen werden.

Komponenten (Software)

Eine Komponente bezeichnet eine Software-Einheit, die ohne Änderung in Software-Systemen verwendet werden kann, die außerhalb der Kontrolle der Entwickler der Komponente liegen. .Eine Software-Komponente stellt i.d.R. ein Element einer komponentenbasierten Anwendung dar und besitzt definierte Schnittstellen, um sie mit anderen Komponenten zu verbinden, sie kann aber auch die Realisierung eines Dienstes bilden.

Middleware

Middleware bezeichnet eine Softwareschicht zwischen dem Übertragungsnetzwerk und den Anwendungen. Ihre Aufgabe ist es, von den Eigenheiten und der Komplexität der verwendeten Infrastruktur zu abstrahieren und den Anwendungen eine reibungslose, standardisierte Interaktion zu ermöglichen (siehe auch http://www.iis.fraunhofer.de/ec/middle/index_d.html).

Monolithen

Als monolithisch wird bezeichnet, was aus einem großen Ganzen besteht, im Gegensatz zu Modularem oder Zusammengesetztem.

.NET

.NET (gesprochen „dotnet“) ist die Bezeichnung für die aktuelle Programmierumgebung von Microsoft. Sie besteht aus verschiedenen Frameworks (Klassenbibliotheken), einer virtuellen Laufzeitumgebung (ähnlich der Java Virtual Maschine) und einer Entwicklungsumgebung (Visual Studio .NET). .NET unterstützt mehrere Programmiersprachen. Dazu gehören C#, C++, J# und Visual Basic.

Offene Schnittstellen

Durch offene, standardbasierte Schnittstellen können Komponenten unterschiedlicher Technologien (CORBA, J2EE, Microsoft .NET) bzw. Produkte unterschiedlicher Hersteller eingesetzt werden. Die medienbruchfreie durchgängige IT-Unterstützung von Geschäftsprozessen bleibt dabei gewährleistet. Dies sichert eine Softwarevielfalt und Nutzer können sich frei für die aus ihrer Sicht beste Lösung entscheiden.

Unabhängig von der eingesetzten Technologie wird sie als „offene Schnittstelle“ bezeichnet, wenn die auszutauschenden Daten, z. B. in Form eines XML-Schemas, sowie die Schnittstelle selbst, im Fall eines Web-Dienstes z. B. in Form einer WSDL-Beschreibung, ausführlich und detailliert dokumentiert sind und diese Dokumentation Jedem frei und kostenlos zur Verfügung gestellt wird.

OSCI (Online Service Computer Interface)

OSCI (Online Service Computer Interface) ist ein deutscher eGovernment-Standard. OSCI wurde speziell für die sichere Übertragung von Daten im eGovernment entwickelt. Zu den wichtigsten Teilen des Standards gehören:

- die sichere und vertrauliche Übertragung von digital signierten Dokumenten über das Internet. Dieser Schritt ist im so genannten OSCI-Transport beschrieben.
- die Standardisierung von Inhaltsdaten, um einen automatischen Datenaustausch zu ermöglichen. In diesem Zusammenhang gibt es mehrere Standardisierungsaktivitäten, die im XÖV (XML für die öffentliche Verwaltung) zusammengefasst werden. Dazu gehören z. B. XMeld, XBau, XJustiz, oder XGewerbe.

Zu der besonderen Charakteristik der OSCI-Transport-Spezifikation gehört die Differenzierung von Zustellinformationen (Absender und Empfänger) und Geschäftsdaten. Wie bei einem „realen“ Postbrief wird zwischen dem eigentlichem Inhalt, der nur für den Absender und Empfänger lesbar ist, und dem Briefumschlag mit Absender- und Empfängerinformationen, die für den Zustelldienst lesbar sind, unterschieden.

OSCI-Transport sieht die vollkommen getrennte Verschlüsselung von Inhaltsinformationen und Zustellinformationen vor. Die Zustellinformationen, zu denen Absender und Empfänger gehören, können nur von einem Zustelldienst bzw. den dazugehörigen Computern gelesen werden. Der Zustelldienst kann nicht auf die Inhalte zugreifen. Die Inhalte können nur vom Absender und vom Empfänger entschlüsselt werden. Auf diesem Weg wird ein zweistufiger „Sicherheitscontainer“ realisiert, der mit den deutschen Gesetzen im Bereich Postzustellung und Postgeheimnis vereinbar ist. Zu den wesentlichen Design-Kriterien der OSCI 1.2 Version gehören:

- Nutzung von offenen Standards wie XML-Encryption, XML-Signaturen oder SOAP,
- Technische Unabhängigkeit, insbes. von Betriebssystem und Programmiersprache,
- Anpassbarkeit bzw. „Skalierbarkeit“ der Sicherheitsmechanismen an unterschiedliche Sicherheitsansprüche.

Plattformunabhängigkeit
Der Grad der Flexibilität, die Ablaufumgebung (Plattform) wechseln zu können, d. h. beispielsweise, dass ein Programm auf verschiedenen Computersystemen mit unterschiedlichen Architekturen, Prozessoren oder Betriebssystemen lauffähig (Portabilität) ist bzw. Informationen mit anderen Programmen über offene Schnittstellen austauschen kann (Interoperabilität).

Portabilität

Als Portabilität wird der Grad der Unabhängigkeit eines Computerprogramms bezeichnet, auf unterschiedlichen Ablaufumgebungen fehlerfrei ablaufen lassen zu können.

SAGA

Standards und Architekturen für eGovernment, ist eine Sammlung von Standards für das deutsche eGovernment. SAGA baut auf Prinzipien des Referenzmodells für Open Distributes Processing (RBM-ODP) aufbaut.

Schichtenarchitektur

Eine Softwarearchitektur, die von mehreren gegeneinander gekapselten Schichten ausgeht, um die Abhängigkeiten der verschiedenen Schichten voneinander zu minimieren und dementsprechend Schichten austauschen zu können. Grund-

figur ist heute die Drei- oder n-Schichten-Architektur (Multi-Tier Architecture). Sie sieht eine Trennung von Präsentation (und Anwender-Interaktion), Logik und Datenhaltung vor. Dabei wird die Präsentation meist mit einem Client, die Logik mit einen Applikationsserver und die Datenhaltung mit einer Datenbank assoziiert. Allerdings werden Drei-Schichten-Architekturen auch gern als Errungenschaft von Web-Anwendungen vorgestellt. In diesem Fall liegt ein erheblicher Teil der Präsentationsschicht nicht auf dem Client, sondern auf einem Server[1]. Man kann diese Trennung der Präsentationsschicht verdeutlichen, indem man die Clients einer eigenen Schicht zuordnet und so zu einer Vier-Schichten-Architektur kommt.[2]

Schnittstelle

Man unterscheidet grundsätzlich zwischen Schnittstellen auf Hardware- und auf Software-Ebene. Auf Hardware-Ebene gibt es unter anderem serielle Schnittstellen wie den Com-Port an dem man z. B. eine Maus anschließen kann und auch parallele Schnittstellen wie z. B. den Drucker-Anschluss. Auf Software-Ebene existieren z. B. Schnittstellen zwischen Programmen, die einen Datenaustausch zwischen diesen ermöglichen.

Serviceorientierte Architektur[3]

Die Bausteine einer serviceorientierten Architektur nehmen meist genau eine von drei möglichen Rollen ein:

- Der Service Provider stellt einen Service zur Verfügung, der eine bestimmte Funktionalität anbietet. Er veröffentlicht den Service bei einem Service Broker.
- Der Service Broker stellt ein Verzeichnis von Services als eigenen Dienst bereit. Er bietet in der Regel Suchfunktionen an, mittels derer Services anhand verschiedener Kriterien gesucht werden können.
- Der Service Requester nutzt einen von einem Service Provider zur Verfügung gestellten Service. Sofern erforderlich, sucht er diesen zunächst bei einem Service Broker.

Shale Framework

Shale ist ein modernes Open-Source-Framework zur Entwicklung von Webanwendungen. Es ist der Nachfolger von Struts und basiert auf der neueren Standard-API (Java Server Faces (JSF)). Der Gebrauch von JSF als fundamentale

---

1 Quelle: E-Government-Handbuch, Abschnitt: Sichere Integration von E-Government-Anwendungen – SIGA, http://www.bsi.de/fachthem/egov/4_siga.htm

2 Siehe Kapitel 6 “Computational Viewpoint: Referenz-Software-Architektur“ in SAGA 2.1 (www.kbst.bund.de/saga)

3 Quelle: E-Government-Handbuch, Abschnitt: Sichere Integration von E-Government-Anwendungen – SIGA, http://www.bsi.de/fachthem/egov/4_siga.htm

UI-Komponententechnologie vereinfacht die Nutzung in eigenen Entwicklungsumgebungen. Gleichzeitig bietet die Shale-Architektur fein gegliederte Dienste und Optionen, die nach den Anforderungen einzelner Anwendungen kombiniert werden können, im Gegensatz zu einem monolithischen Anfrageprozessor der schwierig anpassbar und erweiterbar ist.

Mehr Informationen zum Shale Framework sind unter http://struts.apache.org/-struts-shale zu finden.

Skalierbarkeit

Skalierbarkeit ist die Fähigkeit, Ressourcen so zu erweitern, dass (im Idealfall) die Dienstkapazitäten linear zunehmen. Das wesentliche Merkmal einer skalierbaren Anwendung ist, dass bei steigender Auslastung lediglich zusätzliche Ressourcen und keine umfangreichen Veränderungen der Anwendung selbst erforderlich sind.

SOAP

SOAP (ursprünglich Simple Object Access Protocol) ist ein Protokoll, mit dessen Hilfe Daten zwischen Systemen ausgetauscht und Remote Procedure Calls durchgeführt werden können. SOAP stützt sich auf die Dienste anderer Standards, XML zur Repräsentation der Daten und Internet-Protokolle der Transport- und Anwendungsschicht (vgl. TCP/IP-Referenzmodell) zur Übertragung der Nachrichten. Die gängigste Kombination ist SOAP über HTTP und TCP.

Spring Framework

Spring ist ein JavaBeans-basiertes J2EE Application Framework, das den Umgang mit J2EE-Applikationen erleichtern soll. Der Kern des Spring Frameworks ist ein leichtgewichtiges Komponentenmodell für lokale Middle Tiers, das auf POJOs aufbaut. Dieses ermöglicht einen Verzicht auf die Verwendung von lokalen EJBs und daher eine wesentlich einfachere Entwicklung. Spring verfügt über einen Dependency Injection Mechanismus, ein AOP-Framework sowie ein deklaratives Transaktionsmanagement. Es lässt sich mit JDO und Hilbernate integrieren und bietet MVC-Web-Framework- und JNDI-Unterstützung an.

Mehr Informationen zum Spring Framework sind http://www.springframework.org zu entnehmen.

Struts Framework

Struts ist ein Open-Source-Framework zur Vereinfachung der Entwicklung von Java-Webanwendungen. Es basiert auf dem MVC Design Paradigma, welches die Komponenten Model, View und Controller transparent voneinander trennt. Dem Entwickler stehen dadurch komfortable und einfache Schnittstellen zur Verfügung. Struts bedient sich standardisierter Technologien, wie Java Servlets, Java Server Pages, Ressource Bundles und XML. Verschiedene Modell- (Java Beans, EJB) und Präsentations-Implementierungen (JSP, XML/XSLT) werden

unterstützt. Viele applikationsrelevante Funktionen sind somit bereits implementiert und einsatzbereit.

Mehr Informationen zum Struts Framework sind http://struts.apache.org zu entnehmen.

Tapestry Framework

Tapestry ist ein Open-Source-Framework zur Vereinfachung der Entwicklung von Java-Webanwendungen. Tapestry teilt eine Webanwendung in eine gewisse Anzahl von Seiten, wobei eine Seite aus wieder verwendbaren und konfigurierbaren Komponenten besteht. Durch die klare Trennung von HTML- und Anwendungscode können Java- und HTML-Entwickler gemeinsam an einem Projekt arbeiten. Das Framework wurde so entworfen, dass es einfacher möglich ist, robuste Anwendungen zu erzeugen, die wiederum einfacher zu erstellen, zu debuggen und zu warten sind, als herkömmliche Servlet-Anwendungen. Tapestry lässt sich u.a. mit J2EE, HiveMind und Spring integrieren.

Mehr Informationen zum Tapestry Framework sind zu finden unter http://jakarta.apache.org/tapestry.

TCP/IP

Abkürzung für »Transmission Control Protocol/Internet Protocol«. Das am häufigsten verwendete Protokoll für dezentrale Netzwerkstrukturen. Auch das Internet basiert auf diesem Protokoll bzw. dieser Protokollfamilie.

Wartbarkeit

Die Wartbarkeit von Software gibt an, mit welchem Aufwand und welcher Qualität Änderungen in einem Softwaresystem durchgeführt werden können. Hierbei spielen die geplante Verwendungsdauer der Software, die beteiligten Personen an der Entwicklung und die Verfügbarkeit von Experten eine wichtige Rolle.

Web Service

Ein Web Service

- ist eine abgeschlossene, selbsterklärende und modulare Software-Komponente, die über das Internet veröffentlicht, aufgefunden und benutzt werden kann
- stellt beliebig komplexe Funktionalität zur Verfügung
- kann veröffentlicht und mit einer anderen Anwendung (möglicherweise ebenfalls ein Web Service) zu einem neuen System komponiert werden

Der Nachrichtenaustausch zwischen Web Services basiert zumeist auf dem XML-Format.

Webserver

Ein Webserver ist ein Server im Internet oder Intranet zur Bereitstellung von Web-Seiten und anderen Online-Informationen. Man bezeichnet einen solchen Server als Webserver, da er ausschließlich Informationen für das Web bereitstellt. Dies können HTML Seiten, Textdokumente, aber auch dynamische, datenbankbasierte Seiten sein.

Wiederverwendbarkeit

Wiederverwendbarkeit beschreibt den Grad der Möglichkeit, Modelle, Verfahren, Methoden, Komponenten und andere IT-Systeme in unterschiedlichen Kontexten möglichst unverändert nutzen zu können.

Wirtschaftlichkeit

Gegebenes Maß an Aufgabenerfüllung bei geringst möglichem Mitteleinsatz.

WSDL – Web Service Description Language

Mit Hilfe der Web Service Description Language (WSDL) werden die Schnittstellen, Funktionen, Parameter und Rückgabewerte von Diensten beschrieben. WSDL bietet dabei eine plattform-, programmiersprachen- und protokollunabhängige, XML-basierte Beschreibungsmöglichkeit von Diensten und ihren Schnittstellen.

Mehr Informationen zum Thema WSDL sind http://www.w3.org/TR/wsdl zu entnehmen.

XML – eXtensible Markup Language

Die „eXtensible Markup Language“ (XML) ist ein bewusst sehr einfach und simpel gehaltenes Textformat für den Austausch und die Speicherung von Daten. XML wurde von dem wesentlich umfangreicheren und leistungsfähigeren Format SGML (ISO 8879) abgeleitet. Kernstück des XML-Formats ist die Verknüpfung von Daten mit zugehörigen Bezeichnern (in der Form <Bezeichnung> Daten </Bezeichnung>, Beispiel: <Grundstückspreis> 220.000 </Grundstückspreis>). Aufgrund der Bezeichner (die stets in spitzen Klammern stehen) können Computersysteme die Bedeutung von Daten ermitteln und beispielsweise die Zahlenangaben zum Grundstückspreis und zur Grundstücksfläche in einem Dokument eindeutig ermitteln. Insbesondere durch diese Einfachheit hat XML eine breite Zustimmung erlangt und sich extrem stark verbreitet. XML besitzt mittlerweile eine herausragende Stellung beim Austausch von Daten in allen eCommerce- und eGovernment-Bereichen[1].

[1] Weitere Informationen zum Thema XML sind insbesondere http://www.w3.org/xml zu entnehmen.

XÖV

Unter dem Stichwort „XÖV“ werden die verschiedenen, fachlich orientierten Standards für den interoperablen Datenaustausch im deutschen eGovernment zusammengefasst. Die Koordinierung der verschiedenen Projekte ist eine der Aufgaben der OSCI Leitstelle. Weitere Informationen sind zu finden unter: http://www.osci.de/

XSD – XML Schema Definition

Mit Hilfe von XML Schema Definitionen (XSD) kann die Struktur von XML-Dateien beschrieben werden. Ebenso wie normale XML-Dateien sind auch XSD-Dokumente sowohl von Menschen als auch von Maschinen lesbar. Mit Hilfe der Angaben in XSD-Dateien können sich Computersysteme selbstständig auf die eindeutige Interpretation und Verarbeitung von verschiedenen XML-Datensätzen einstellen[1].

XSL – eXtensible Stylesheet Language

Die eXtensible Stylesheet Language (XSL) besteht aus zwei Teilen. Eine Sprache (XSLT), die die Transformation von unterschiedlichen XML-Datensätzen ermöglicht und einem Vokabular (XSL-FO) zur Formatierung von XML-Dateien. XSL-FO kann beispielsweise genutzt werden, um spezielle Layouts einer XML-Datei für verschiedene Medien zu erzeugen. Mit XSL-FO ist es z. B. möglich, ein Layout zum Drucken und ein Layout für die Darstellung am Bildschirm zu verwenden. XSLT kommt beispielsweise dann zum Einsatz, wenn XML-Datenstrukturen verschiedener Software-Anwendungen unterschiedlich aufgebaut sind, obwohl sie im Grunde gleiche Objekte beschreiben. Ein XML-Datensatz zur Beschreibung eines Grundstücks könnte für die Grundstücksfläche z. B. den Bezeichner <Grundstücksfläche> verwenden. Eine andere Anwendung die davon ausgeht, dass für die Grundstücksfläche der Bezeichner <Fläche> verwendet wird, könnte den XML-Datensatz dann nicht mehr richtig interpretieren. Mit Hilfe der Zuordnungsvorschriften in XSLT-Dateien können solche Datensätze in das jeweils gewünschte Format transformiert werden[2].

[1] Weitere Informationen zu diesem Thema sind insbesondere http://www.w3.org/xml/schema zu entnehmen.

[2] Mehr Informationen zum Thema XSL sind insbesondere http://www.w3.org/TR/xsl zu entnehmen.

A.4 Tabellenverzeichnis

A.5 Abbildungsverzeichnis