Erarbeitung und Evaluierung von möglichen Lösungsarchitekturen für das System „Besondere Nutzung Luftraum“

Diplomarbeit

zur Erlangung des akademischen Grades
Diplominformatiker

Humboldt-Universität zu Berlin
Mathematisch-Naturwissenschaftliche Fakultät
Institut für Informatik

eingereicht von: Volker Grabsch

Gutachter(innen): Prof. Dr. sc. nat. Klaus Bothe
Dr. Kai Renz

eingereicht am: 11. Juni 2014 verteidigt am: 10. Juli 2014

Zusammenfassung

Eine wichtige Aufgabe der Flugsicherung ist die Abwicklung von besonderen Vorhaben wie zum Beispiel Fallschirmsprüngen, Fotoflügen oder Luftfahrtveranstaltungen. Deren Bearbeitung von der Antragstellung bis zur Durchführung wird derzeit mit Standardsoftware und Papierausdrucken realisiert. Diese Arbeitsweise stößt an ihre Grenzen, daher wird eine verbesserte Softwareunterstützung gewünscht.

In einer vorangegangenen Arbeit wurden bereits die Arbeitsabläufe analysiert und daraus funktionale Anforderungen abgeleitet. In dieser Diplomarbeit wird von Seiten der Software-Architektur untersucht, wie solch eine Software aufgebaut sein könnte, und inwieweit diese sinnvoll in das bestehende System STANLY_ACOS integriert werden kann, das eine ähnliche Aufgabe erfüllt und als mögliche Basis zur Verfügung steht.

Dazu wird zunächst eine zweite, eigene Anforderungserhebung durchgeführt, die die erste verfeinert und ergänzt. Darauf basierend werden mögliche Architekturen entworfen, gefolgt von einer Evaluation, in der die am besten geeignete Architektur ausgewählt wird. Diese wird zum Schluss validiert, indem sie in Form eines Prototypen umgesetzt wird.

Die Kernpunkte der ursprünglichen Anforderungen sind, dass alle Beteiligten auf einen gemeinsamen Informationsstand zugreifen können, und dass alle Vorhaben in einer gemeinsamen Karte visualisiert werden. Die zweite, eigene Erhebung verfeinert diese mit Schwerpunkt auf nichtfunktionale Anforderungen, etwa, wie die Netzwerkstruktur der Server und Arbeitsplatzrechner aussieht, und welche Echtzeit-Anforderungen es gibt.

Auf dieser Basis werden systematisch Softwarearchitekturen entworfen. Dabei werden die Hauptkomponenten identifiziert sowie alle Teilmengen und Kommunikationsstrukturen betrachtet. In einheitlicher Notation werden jeweils verschiedene Standardkomponenten und Arten von Eigenentwicklungen analysiert, die auf 4 sinnvolle Strukturen und 78 funktionierende Komponenten-Kombinationen reduziert werden. All diese Architekturkandidaten werden gemeinsam evaluiert. Hierzu werden sie anhand eines Kriterien-Kataloges verglichen, der sich wiederum größtenteils aus den Anforderungen ergibt. Die Entscheidung fällt auf die intern als BS-IS/E-SX-ER-ER bezeichnete Architektur.

Diese Architektur wird als Prototyp umgesetzt, der zwar nicht alle Anforderungen erfüllt, aber zumindest diejenigen, die in Bezug auf die Architektur eine gewisse Herausforderung darstellen. Die erhofften Synergie-Effekte durch Realisierung als STANLY_ACOS-Modul treten überwiegend ein, auch wenn einige Funktionalitäten von STANLY_ACOS für den Prototyp nicht direkt einsetzbar sind und teilweise reimplementiert werden müssen. Zudem wartet das verwendete Framework *ExtJS* mit einigen besondere Herausforderungen auf. Nichts davon erfordert jedoch ein Redesign, oder gar eine Abweichung von der Architektur.

Der Prototyp bestätigt damit voll und ganz die Tragfähigkeit der Architektur und ist zudem zu einer präsentierfähigen Diskussions-Grundlage geworden.

Danksagung

Während der Erstellung dieser Diplomarbeit wurde ich von vielen Menschen unterstützt, denen ich dafür sehr dankbar bin.

Mein besonderer Dank gilt Dr. Kai Renz für seine umfassende Unterstützung in allen Phasen dieser Arbeit, für sein klares Feedback, seine pragmatischen Einschätzungen und vor allem dafür, dass er mich immer wieder motivierte, trotz aller Ablenkungen die Arbeit Stück für Stück voranzubringen.

Ebenfalls danke ich Prof. Dr. Klaus Bothe, der mir stets genau die richtigen Fragen stellte und gerade in schwierigen Situationen jederzeit mit gutem Rat zu helfen wusste.

Für die solide Vorarbeit danke ich Nils Hartwig, auf dessen Schultern ich mit dieser Diplomarbeit stehe.

Herzlichen Dank auch an Michael Stumpe, der sich stets die Zeit dafür nahm, meine vielen Fragen zum Thema *Besondere Nutzung Luftraum* geduldig zu beantworten.

Ich danke Mark Kleinschmidt, Wolfgang Diehl, Ashley Williams, Ralf Hortmann und den vielen anderen DFS-Mitarbeitern, die mich unbürokratisch unterstützten und mir spannende Einblicke in verschiedenste Bereiche der Flugsicherung gaben. Auch wenn ich bei weitem nicht all ihre Impulse in diese Arbeit einbringen konnte, so vermittelten sie mir doch einen guten Überblick und halfen mir bei der besseren Einordnung meiner Arbeit in das große Ganze.

Ich danke Matthias Pohl, Angelika Grabsch, Mario Krell und den vielen anderen Menschen, denen ich Entwürfe meiner Arbeit zeigen konnte und die mir wertvolles Hinweise auf auf Tippfehler und ungeschickte Formulierungen gaben.

Nicht zuletzt danke ich meiner Lebensgefährtin Antje Diels wie auch meiner gesamten Familie für ihre uneingeschränkte Unterstützung in dieser arbeitsreichen Zeit.

Selbständigkeitserklärung

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbständig verfasst und nur unter Verwendung der angegebenen Quellen und Hilfsmittel angefertigt habe. Weiterhin erkläre ich, eine Diplomarbeit in diesem Studiengebiet erstmalig einzureichen.

Berlin, den 11. Juni 2014 _____

Inhaltsverzeichnis

1. Einführung

1.1. Flugsicherung

„Flugsicherung dient der sicheren, geordneten und flüssigen Abwicklung des Luftverkehrs.“[1] Doch wie wird Flugsicherung in Deutschland gewährleistet? Bekanntlich stemmen die Fluglotsen den wichtigsten Teil dieser Arbeit. Sie haben stets einen genauen Überblick über ihren jeweiligen Luftraum, sehen dort sämtliche Flugzeug-Bewegungen und geben Anweisungen direkt an die Piloten.

Doch das allein wäre ein äußerst unvollständiges Bild der Flugsicherung. Um die Fluglotsen herum ragt eine großes Unternehmen, die DFS Deutsche Flugsicherung GmbH, die den Fluglotsen in jeder erdenklichen Form zuarbeitet. Zum einen muss die DFS für einen reibungslosen organisatorischen Ablauf sorgen. Alles, was sich an Planungsarbeit im Vorfeld erledigen lässt, wird von entsprechenden Abteilungen der DFS übernommen. So können sich die Fluglotsen am Ereignistag auf das Wesentliche konzentrieren. Zum anderen muss die DFS für eine zuverlässige Technik sorgen und diese ständig weiterentwickeln. Der Bereich *Besondere Nutzung Luftraum* ist ein typisches Beispiel für dieses Zusammenspiel aus organisatorischer und technischer Zuarbeit.

Die DFS ist eine GmbH und damit ein privatrechtlich organisiertes Unternehmen. Sie liegt zu 100% in Bundeseigentum und finanziert sich hauptsächlich aus Gebühren, die beispielsweise Fluggesellschaften für die Dienstleistungen der DFS bezahlen. Im Jahr 2012 arbeiteten 6100 Mitarbeiter für die DFS und 244 Menschen begannen dort eine Ausbildung. Etwa die Hälfte der Mitarbeiter arbeiten direkt in der Flugverkehrskontrolle (2012 waren es 48%), die andere Hälfte arbeitet an der operativen Technik sowie in unterstützenden Bereichen. [DFS13a]

Die Fluglotsen sind in verschiedenen Bereichen tätig: in der Platzkontrolle (Tower), in der An- und Abflugkontrolle (Approach) sowie in der Bezirkskontrolle (Center). Letztere sind für das Thema dieser Arbeit wichtig. Diese *Center-Lotsen* arbeiten in Radarkontrollzentralen (Center[2]), das heißt, sie arbeiten nicht auf Sicht, sondern nehmen den Luftraum ausschließlich über entsprechende Radar- und Informationssysteme wahr.

[1] Definition aus [Deu12, LuftVG, §27c Abs. 1]

[2] Genau genommen gibt es *ACC (Area control centre)*, die den unteren Luftraum kontrollieren, und *UAC (Upper area control centre)*, die den oberen Luftraum kontrollieren. Da diese Unterscheidung hier jedoch nicht relevant ist, wird in dieser Arbeit der zusammenfassende Begriff *Center* verwendet.

Abbildung 1.: Unterteilung des deutschen Luftraums in Sektoren
(Karte aus STANLY_ACOS FABEC)

Der gesamte deutsche Luftraum wird derzeit durch fünf Center überwacht, von denen sich vier in Deutschland befinden.[3] Jedes Center hat einen definierten Luftraum zu verwalten, den es zur besseren Handhabung in sogenannte *Sektoren*[4] unterteilt, die während eines geringen Verkehrsaufkommens zu *Sektorgruppen* zusammengefasst werden.

Jede Sektorgruppe wird von jeweils zwei Lotsen gemeinsam bearbeitet, die zusammen an einem Arbeitsplatz sitzen, aber unterschiedliche Aufgaben erfüllen[5]. Einer Gruppe von Lotsen sind jeweils ein oder zwei *Supervisor* übergeordnet. Die Supervisor arbeiten im gleichen Kontrollraum wie die Lotsen, und müssen stets einen Überblick über den gesamten Luftraum haben. Sie sind vor allem für die kurz- und langfristige Planung der Luftraum-Nutzungen verantwortlich, und stellen die letzte Instanz für Genehmigungen und Absagen dar.

Dabei erhalten die Supervisor intensive Zuarbeit durch verschiedene Abteilungen. Diese befinden sich ebenfalls im Center, allerdings außerhalb des relativ stark isolierten Kontrollraums. Eine Abteilung ist für die *Besondere Nutzung Luftraum* zuständig, um die es im folgenden Abschnitt geht.

1.2. Besondere Nutzung Luftraum (BNL)

Die Luftverkehrs-Ordnung [Bun12, LuftVO] legt die Verkehrsregeln in der Luft fest.[6] Unter anderem stellt sie klar, welche Aktivitäten in der Luft eine vorherige Erlaubnis benötigen, und durch wen diese Erlaubnis jeweils ausgesprochen werden kann. Für viele Aktivitäten ist die Entscheidungsinstanz einfach die jeweilige Landes-Luftfahrtbehörde [Bun12, LuftVO, §15, §15a, §16].

Es gibt jedoch auch Aktivitäten, die direkt durch die zuständige Flugverkehrskontrollstelle genehmigt werden. Diese sind aufgeführt unter *Besondere Benutzung des kontrollierten Luftraums* [Bun12, LuftVO, §16a], was üblicherweise zu *Besondere Nutzung Luftraum* abgekürzt wird, oder einfach BNL.

Die LuftVO lässt den Flugverkehrskontrollstellen bestimmte Ermessensfreiräume. Daher gibt die DFS noch einmal genauere Anordnungen in ihrem amtlichen Mitteilungsblatt *Nachrichten für Luftfahrer* (NfL) bekannt. Die für BNL-Aktivitäten relevanten Ausgaben der NfL sind:

[3] Nach Stand von 2013 sind dies: EDUU in Karlsruhe, EDWW in Bremen, EDGG in Langen und EDMM in München. Eine Sonderstellung nimmt das Center EDYY in Maastricht ein, welches nicht von der DFS, sondern von der europäischen Flugsicherung EUROCONTROL betrieben wird und unter anderem den oberen Luftraum von Hannover überwacht.

[4] Diese sind entgegen ihrem Namen in der Regel keine geometrischen (Kreis-)Sektoren.

[5] Radarlotse und Koordinationslotse, für die es sogar separate Zulassungen gibt

[6] Die rechtliche Bindung der LuftVO ist im Luftverkehrsgesetz [Deu12, LuftVG] festgeschrieben: In §31 wird das *Bundesministerium für Verkehr, Bau und Stadtentwicklung* mit allen Bundes-Aufgaben des Luftverkehrs betraut, verbunden mit dem Recht, diese Aufgaben teilweise zu delegieren. Die nachfolgenden Paragraphen §31a, §31b, §31c, §31d, §31e, §31f, §32 und §32a stecken den genauen Rahmen ab, in welchem das Ministerium ermächtigt wird, eigene Rechtsverordnungen zu erlassen. Davon macht es Gebrauch, indem es die Luftverkehrs-Ordnung [Bun12, LuftVO] herausgibt und von Zeit zu Zeit aktualisiert.

- NfL I 110 / 11 – Bekanntmachung der besonderen Voraussetzungen für die Erteilung von Flugverkehrskontrollfreigaben (ersetzt NfL I 332 / 02) [DFS11]
- NfL I 59 / 07 – Voraussetzungen für die Erteilung einer Flugverkehrskontrollfreigabe zur Durchführung von Fallschirmabsprüngen und zum Abwerfen von Gegenständen an Fallschirmen im kontrollierten Luftraum [DFS07]
- NfL I 32 / 09 – Bekanntmachung über die Festlegung von Mindestvorlaufzeiten für die „Besondere Nutzung Luftraum“ [DFS09]

Auf ziviler Seite gibt es in den entsprechenden Centern jeweils ein BNL-Büro. Darüber hinaus gibt es militärische BNL-Büros, die mit den zivilen bei Bedarf eng kooperieren. In dieser Arbeit geht es hauptsächlich um die zivile Seite. Im Fokus dieser Arbeit steht das Center in Langen als geplanter erster Nutzer der Software.

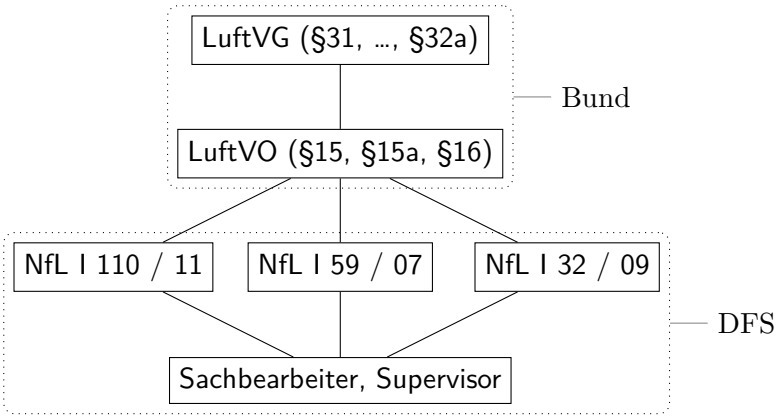


Abbildung 2.: Regelungen und Entscheidungsträger

Die typischen zivilen BNL-Aktivitäten sind:

- Kunstflug (Aerobatic)
- Fallschirmsprung (Parachute Jumping Exercise)
- Gasausblasung (Gas Blowout)
- Bombenentschärfung (Bomb Disposal)
- Luftfahrtveranstaltung (Airshow)
- Segelflug (Gliding)
- Fotoflüge (Scanning Flights)

Darüber hinaus gibt es weitere, seltenere BNL-Aktivitäten, die unter *Sonstige (Others)* zusammengefasst werden.

Die Antragsteller auf ziviler Seite sind typischerweise die Polizei (Gasausblasung, Bombenentschärfung) sowie private Unternehmen (Fotoflüge, Kunstflug und anderes). Bei militärischen Vorhaben ist der Antragsteller die COMIL[7]. Da die Bundeswehr ein eigenes BNL-Büro betreibt, findet in diesem Fall eine engere Kooperation als mit anderen

[7]Koordinationszentrale für die militärische Luftraumnutzung (Coordination Center for Military Airspace Utilization) [Amt13]

Antragstellern statt. Soll beispielsweise ein militärischer Fallschirmsprung in einem zivilen Luftraum stattfinden, so reiht sich die COMIL ganz normal neben die zivilen Antragsteller ein, kann genauso eine Genehmigung oder Absage erhalten, übernimmt aber im Falle der Genehmigung einige Verwaltungsaufgaben wie die Erstellung des NOTAM[8].

Wann immer möglich werden BNL-Vorhaben langfristig beantragt. Die Anträge werden beim BNL-Büro eingereicht, das sie sammelt und vorab überprüft. Dabei achten die Sachbearbeiter nicht nur auf die formale Korrektheit der Anträge, sondern vor allem auf mögliche Konflikte mit bereits geplanten Vorhaben – BNL-Vorhaben wie auch anderen Vorhaben. Ein Konflikt entsteht, wenn der Zeitbereich, das geografische Gebiet und der Höhenbereich zu nah an einem anderen Vorhaben liegen. Die genehmigten Vorhaben reichen die Sachbearbeiter an den jeweils diensthabenden Supervisor[9] weiter. Zudem prüfen die Sachbearbeiter vorab, welche Sektoren betroffen sind, sodass der Supervisor schnell feststellen kann, welche Lotsen sich um das Vorhaben kümmern müssen.

Daneben gibt es auch kurzfristige BNL-Vorhaben, die keinen Vorlauf durch das BNL-Büro erfahren. Diese Anträge gehen direkt beim diensthabenden Supervisor ein, der nun im laufenden Betrieb sämtliche Überprüfungen selbst durchführen muss, um kurzfristig die richtigen Entscheidungen zu treffen.

Doch auch bei langfristigen BNL-Vorhaben muss der Supervisor kurz vor der Durchführung zur Sicherheit noch einmal eine Konfliktprüfung gegen die aktuelle Lage im Luftraum durchführen, da diese von der ursprünglich geplanten stets abweichen kann. Notfalls muss der Supervisor ein Vorhaben kurzfristig absagen, er hat dabei stets das letzte Wort. Natürlich ist es Ziel der Planung, solch eine Situation im Vorfeld zu vermeiden.

Alle Informationen zu den laufenden BNL-Vorhaben müssen zwischen verschiedensten Akteuren übergeben werden, was nicht immer reibungslos und ohne Medienbrüche vonstatten geht. Wie bereits erklärt, gelangen die Informationen zunächst vom Antragsteller über das BNL-Büro zum Supervisor, und von diesem zu den entsprechenden Lotsen. Darüber hinaus übergibt der Supervisor beim Schichtwechsel die Informationen an seinen Nachfolger, der sich dann erneut ein Bild von der Lage machen muss. Je nach Vorhaben müssen der Supervisor und das BNL-Büro diverse externe Stellen benachrichtigen, unter anderem durch Erstellen eines NOTAM. Je besser und einheitlicher die Informationen aufbereitet sind, desto weniger aufwändig sind die Übergaben.

Zusammengefasst gibt es die folgenden kritischen Stellen im BNL-Arbeitsablauf:

- Konfliktprüfung
  - im Vorfeld durch das BNL-Büro
  - am Tag der Durchführung durch den Supervisor
- Übergabe der Informationen
  - vom Antragsteller an das BNL-Büro oder direkt an den Supervisor
  - vom BNL-Büro an den Supervisor

[8] Benachrichtigung an alle Luftfahrer über die Sperrung des Luftraums (Notice to Airmen)

[9] Da es keine allgemein verwendete weibliche Form des Wortes *Supervisor* gibt, wird in dieser Arbeit konsequent die männliche Form verwendet.

– vom Supervisor an die Lotsen
– vom Supervisor an den nächsten Supervisor (Schichtwechsel)
– an externe Stellen

Leider fehlt an genau diesen kritischen Stellen eine direkt auf die Arbeitsabläufe zugeschnittene Software-Unterstützung. Zwar arbeitet das BNL-Büro intensiv mit Standardwerkzeugen wie Tabellenkalkulation, GIS/CAD-Tools, stellt Informationen über Netzwerkordner und E-Mail bereit, und pflegt wichtige Lufträume in das Informationssystem ATCAS der Lotsen ein. Doch das Zusammenspiel dieser einzelnen Werkzeuge hat seine Grenzen.

Die Probleme beginnen schon bei der Antragstellung. Allein die Spezifikation des betroffenen Luftraums erfolgt auf unterschiedlichste Arten und Weisen. Einige skizzieren den Luftraum von Hand auf einer gedruckten ICAO-Karte[10], andere senden OVL-Dateien [Cac10], die direkt in die gängigen GIS/CAD-Tools eingeladen werden können. Einige dieser Lufträume gibt das BNL-Büro in das ATCAS-System ein, sodass sie von den Lotsen bei Bedarf direkt in die Radardaten eingeblendet werden können. Dies ist jedoch mit einigem Aufwand verbunden, da ATCAS ein operatives System ist und daher keine direkte Netzwerkverbindung zur Außenwelt hat, nicht einmal zum BNL-Büro.[11] Daher lohnt sich der Aufwand nur für sehr einfach beschreibbare Lufträume bzw. solche, die in nächster Zeit sehr oft verwendet werden. Alle anderen Lufträume landen in ausgedruckter Form beim Supervisor, der sie – immer noch in Papierform – an die Lotsen weiter gibt. So muss der Supervisor letztendlich sämtliche Konfliktprüfungen mental durchführen, mit Blick auf Bildschirm und auf verschiedene Kartenausdrucke in unterschiedlicher Qualität und manchmal sogar unterschiedlichen Kartenprojektionen. Die Lotsen stehen danach vor dem gleichen Problem, wenn auch in abgeschwächter Form. Neben der Luftraum-Spezifikation müssen auch viele andere Informationen weiter gegeben werden, die sich im Laufe des Tages ändern können. Wenn der Supervisor kurzfristige Änderungen erfährt, erlauben die bestehenden Tools keine schnelle Dokumentation. So muss er sich handschriftliche Notizen machen und dieser später nachtragen. Er darf nicht vergessen, alle weiteren betroffenen Stellen zu informieren, sonst kann es zu unterschiedlichen Informationsständen kommen.

Dieser Arbeitsablauf verlangt eine sehr hohe Konzentration von allen Beteiligten und ist weit von dem entfernt, was technisch möglich ist.

Natürlich geschieht es immer wieder in einem so großen Unternehmen wie der DFS, dass ein historisch gewachsener Arbeitsprozess einer Überarbeitung bedarf. Arbeitsschritte müssen besser visualisiert, Informationen synchron gehalten und unnötige Belastungen wegautomatisiert werden. Die entsprechenden Software- und manchmal auch Hardware-Entwicklungen organisieren dann andere Abteilungen. In diesem konkreten Fall wurde die Abteilung *ATM Daten und Dienste* (OA/L) mit der Software-Entwicklung beauftragt.

[10] Die ICAO-Karte ist die von der DFS offiziell herausgegebene Luftfahrtkarte, die den Richtlinien der International Civil Aviation Organization genügt. Ein beispielhafter Ausschnitt aus der ICAO-Karte befindet sich unter [DFS12].

[11] Eine genauere Beschreibung der Netzwerkstruktur befindet sich in Anforderung 41.

So ist der Bereich *Besondere Nutzung Luftraum* ein typisches Beispiel für das in Abschnitt 1.1 hervorgehobene Zusammenspiel von organisatorischer und technischer Zuarbeit für die Supervisor und Fluglotsen: Das BNL-Büro nimmt ihnen viel organisatorische Arbeit mit den BNL-Vorhaben ab. Die Abteilung OA/L steuert die Software-Entwicklung, sodass BNL-Vorhaben in Zukunft einen geringeren Aufwand verursachen.

1.3. Flexible Luftraumnutzung (FUA)

Im Jahr 2004 führten das Europäisches Parlament und der Rat der Europäischen Union im Rahmen der *Interoperabilitäts-Verordnung* das Konzept der flexiblen Luftraumnutzung (FUA, Flexible Use of Airspace) für alle Mitgliedstaaten verbindlich ein. [Eur04, S. 10] Ende 2005 präzisierte die Europäische Kommission dieses FUA-Konzept durch eine weitere Verordnung. [Eur05] Es gibt inzwischen sogar ein von der europäischen Flugsicherung EUROCONTROL produziertes Erklärungsvideo zu FUA. [EUR09]

Ziel von FUA ist es, die bis dahin ausschließlich militärisch genutzten Lufträume auch dem zivilen Flugverkehr zugänglich zu machen. Die starre Trennung zwischen militärischen und zivilen Lufträumen ist nicht mehr zeitgemäß und wird aufgelöst. Stattdessen gibt es einen gemeinsamen Pool von Lufträumen, die je nach Bedarf den zivilen wie auch militärischen Nutzern zeitweise zur Verfügung gestellt werden. Dies ist ein wichtiger Schritt, um die immer knapper werdene Ressource *Luftraum* so effizient wie möglich zu verteilen.

Die meisten zivilen Nutzer, besonders der kommerzielle Passagierflug, bewegen sich entlang eines Pfades in einem großen Flugrouten-Netz (Abbildung 3). Die militärischen Nutzer hingegen benötigen große Lufträume, die weite Gebiete abdecken. Dieses sind meist Trainingsgebiete für komplexe Flugmanöver. Einige sind jedoch auch reine Sperrgebiete, in denen nichts fliegen darf, weil beispielsweise Schießübungen am Boden oder auf See stattfinden.

Diese unterschiedlichen Nutzungsarten führen im Wesentlichen zu zwei Arten von Konflikten: Die einen Konflikte treten auf, wenn mehrere Parteien die großen Lufträume zu überlappenden Zeiten in überlappenden Hohen nutzen wollen. Die anderen Konflikte entstehen dort, wo die Flugrouten die Lufträume kreuzen oder diesen sehr nahe kommen. Letztere werden normalerweise so gelöst, dass entweder ein kompletter Routenabschnitt gesperrt und damit gar nicht beflogen wird (Abbildung 4). Es kommt jedoch auch vor, dass eine Flugroute nicht gesperrt wird, und die sich auf der Flugroute befindenen Flieger aufwändig um den betroffenen Luftraum herum geleitet werden.

Das FUA-Konzept wird in drei Level unterteilt, die aufeinander aufbauen:[12] [EUR12c]

FUA Level 1 (strategisch / strategic)

Die Flugsicherung der jeweiligen Nation trifft entsprechende Vereinbarungen mit den zivilen und militärischen Luftraumnutzern. Sie etabliert die notwendigen internen Verfahren und definiert dabei die Lufträume, in denen FUA zum Einsatz

[12] *Strategisch* ist alles, was bis ungefähr 2 Tage vor dem Ereignistag geschieht. *Prätaktisch* ist alles, was 1–2 Tage vor dem Ereignistag geschieht. *Taktisch* ist alles, was am Ereignistag geschieht.

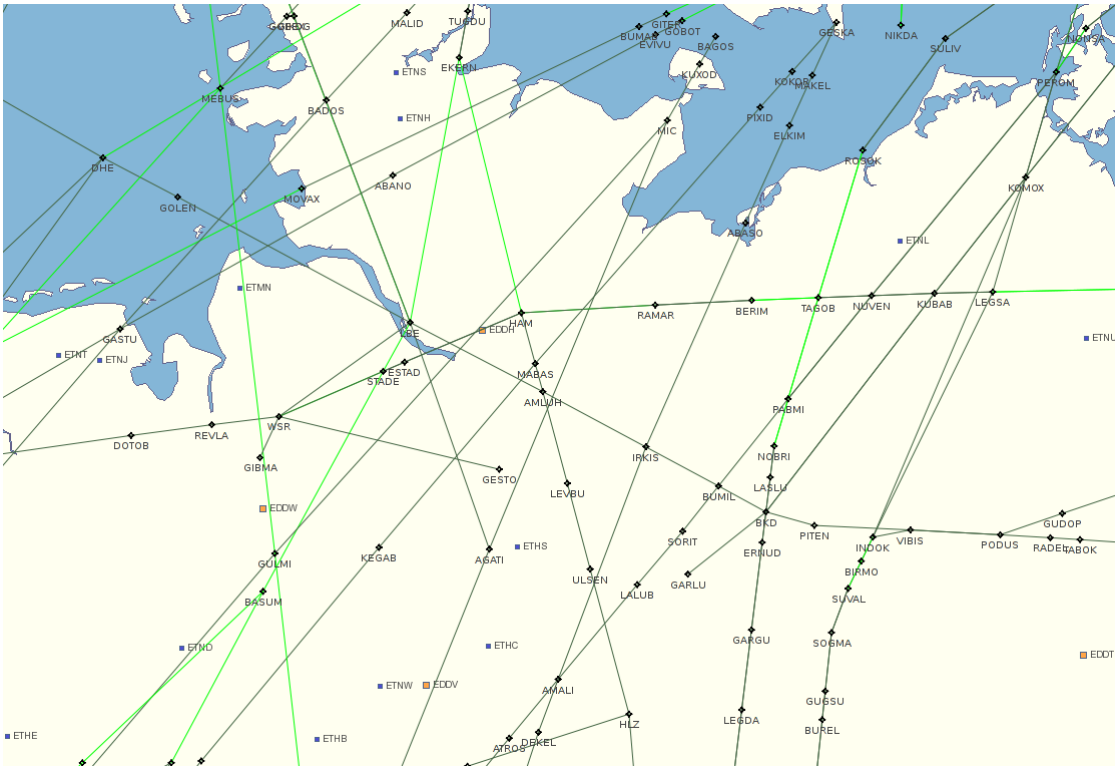


Abbildung 3.: Auszug einiger Flugrouten und ihrer Wegpunkte im Norden Deutschlands (Karte aus STANLY_ACOS FABEC)

kommen wird. Wo nötig, errichtet sie neue Lufträume – auch grenzübergreifende, die sie mit den jeweiligen Nachbarländern koordiniert.

FUA Level 2 (prätaktisch / pre-tactical)

Die Flugsicherung etabliert eine sogenannte Luftraummanagementzelle (AMC, Airspace Management Cell), die sich um das Tagesgeschäft der Luftraum-Zuteilung kümmert. Die AMC ist gemeinsamen zivil/militärisch besetzt und mit allen Kompetenzen ausgestattet, um zeitnah verbindliche Entscheidungen zu treffen.[13] [EUR12a, S. 76 ff] Sie setzt Prioritäten und löst Konflikte zwischen allen Nutzungswünschen, militärischen wie zivilen.

Jeden Abend gibt sie einen Luftraumnutzungsplan (AUP, Airspace Use Plan) heraus, der für den nächsten Tag angibt, wer wann welchen Teil von welchem Luftraum in welchen Höhen nutzen darf, und welche Abschnitte im Flugrouten-Netz in welchen Zeiten und Höhen gesperrt oder passierbar sind. Falls nötig, kann sie am jeweils nächsten Tag, dem Ereignistag, einmal oder mehrmals einen aktualisierten

[13] Die Entscheidungskompetenz beschränkt sich natürlich auf die (lang- und kurzfristige) Planung. Das letzte Wort hat – wie immer – der diensthabende Supervisor in dem für den Luftraum zuständigen Center. Diese kurzfristigen Absagen (refusals) sind jedoch ziemlich selten.

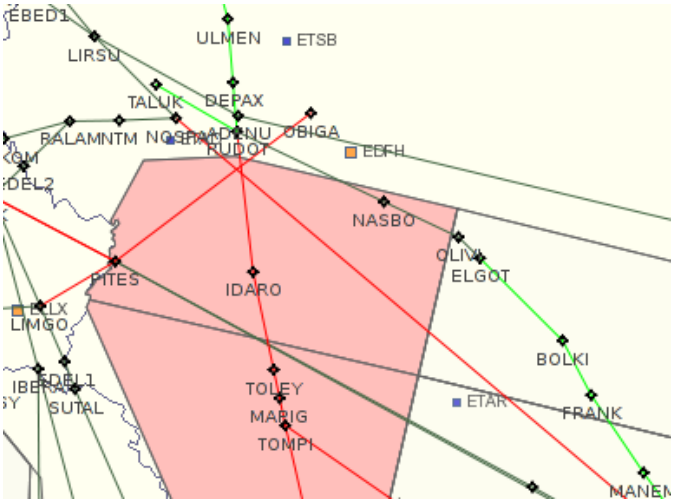


Abbildung 4.: Aktiviertes Beschränkungsgebiet (Restricted Area). Die dadurch betroffenen und gesperrten Flugrouten sind rot dargestellt. (Karte aus STANLY_ACOS FABEC)

Luftraumnutzungsplan (UUP, Updated Airspace Use Plan) herausgeben.[14]

FUA Level 3 (taktisch / tactical)

Die Flugsicherung ermöglicht es den Luftraumnutzern, die per AUP/UUP zugewiesenen Lufträume in Echtzeit[15] abzusagen, sowie zeitlich und räumlich abzuändern. Wird dadurch ein Flugrouten-Abschnitt frei, leiten die Lotsen den Flugverkehr kurzfristig um, sodass die gerade entstandene Abkürzung auch wirklich genutzt wird. Genauso ermöglicht es die Flugsicherung, freigewordene Lufträume kurzfristig wieder zu belegen, wenn es einen Luftraumnutzer gibt, der auf solch eine günstige Gelegenheit gerade wartet.

All das erfordert eine Echtzeit-Koordination zwischen der AMC, den Centern und der Flugverkehrfluss-Kontrolle (Flow Management), und damit einen ständigen, möglichst automatisierten, Informationsabgleich aller beteiligten Systeme.

Da das FUA-Konzept im Rahmen des Programms *einheitlicher europäischer Luftraum* (SES, (Single European Sky) [Eur12d] entstanden ist, gibt es zu jedem einzelnen FUA-Level auch noch Anforderungen an die europäische Zusammenarbeit der Nationen, was jedoch an dieser Stelle zu weit führen würde.

Zwar hat FUA mit BNL erst einmal nichts zu tun, doch beide haben große Ähnlichkeiten in ihren Anforderungen an eine Software-Unterstützung: Lufträume sollen für

[14] Genau genommen könnten die UUPs bereits als taktisch und nicht als prätaktisch gelten, da sie direkt am Ereignistag herausgegeben werden.

[15] Der Begriff *Echtzeit* hat in diesem Zusammenhang nicht die in der Informatik übliche Bedeutung (Zeiträume von Zehntelsekunden und weniger), sondern bedeutet, dass Planungs- und Abstimmungs-Prozesse, die üblicherweise Stunden bis Tage in Anspruch nehmen, nun innerhalb von Sekunden bis wenigen Minuten ablaufen.

bestimmte Zeiten und bestimmte Höhen angefragt werden, langfristig wie auch kurzfristig, alle denkbaren Konflikte sollen erkannt und von Koordinierungsstellen gelöst werden, und alle beteiligten Stellen (insbesondere die Supervisor) müssen zu jeder Zeit den aktuellen Stand sehen können.

Daher ist es naheliegend, zu prüfen, inwieweit sich das bestehende FUA-System namens *STANLY_ACOS* um die für BNL benötigten Funktionalitäten erweitern lässt. Abgesehen von auf BNL zugeschnittenen Eingabemasken (Anforderung 1) fehlt es STANLY_ACOS vor allem an der Fähigkeit, Luftraum-Geometrien ad hoc zu definieren (Anforderung 6), sowie Anträge von außerhalb des abgeschotteten Intranets sicher entgegen zu nehmen (Anforderung 14).

1.4. STANLY_ACOS

Die DFS begann schon sehr früh mit der Umsetzung von FUA und durchlief dabei mehrere Entwicklungsphasen, die im Folgenden beschrieben werden. [Hor13]

STANLY_VPA und STANLY_MVPA – Bereits 2002 gab die Abteilung *ATM Daten und Dienste* (OA/L)[16] die Entwicklung eines Prototypen namens STANLY_VPA in Auftrag, der 2004 durch eine neue Applikation STANLY_MVPA abgelöst wurde.[17] Dabei ging es zunächst nur um einen einzigen Luftraum im Nordosten Deutschlands, der von einer TRA (Temporary Restricted Area) zur ersten MVPA (Military Variable Profile Area) weiterentwickelt wurde. Der Luftraum wurde hierzu schachbrettartig unterteilt, wie in Abbildung 5 gezeigt, sodass die Nutzer jeden Teil-Luftraum einzeln oder in Kombination mit anderen Teilen buchen konnten. Die Größe der „Schachbrettfelder“ wurde mit der Zeit an den tatsächlichen Bedarf angepasst. [DFS08a] [DFS08b]

STANLY_ACOS 1 – Über die Jahre kamen weitere Lufträume hinzu, die jedoch nicht in MVPAs umgewandelt wurden. STANLY_MVPA entwickelte sich von einem reinen MVPA-System zu einem allgemeinen Luftraumverwaltungs-System, das 2008 in STANLY_ACOS (Airspace Coordination System) umbenannt wurde. Mit STANLY_ACOS können Lufträume je nach Bedarf auf allen drei FUA-Leveln betrieben werden.

Dieses System wird inzwischen als STANLY_ACOS 1 bezeichnet, um es von den nachfolgenden Systemen zu unterscheiden.

STANLY_ACOS 1 basiert nach wie vor auf einer klassischen 3-Schichten-Architektur[18], bestehend aus Persistenzschicht, Mittelschicht und Client-Schicht, deren Mittelschicht stark an das Muster Model-View-Presenter (MVP) angelehnt ist, ergänzt um ein Messaging-System, das server-initiierte Kommunikation mit den

[16] Damals hieß diese Abteilung noch *Lage- und Informationszentrum* (LIZ), später CC/FZ, dann OA/L.

[17] Diese wurden in der Abteilung OA/L als Teil der STANLY-Produktfamilie (Statistics and Analysis System) konzipiert, daher jeweils der vordere Namensteil *STANLY_*.

[18] Dass STANLY_ACOS 1 als Webanwendung aus den frühen 2000er Jahren überhaupt ein wiederverwendbares Framework als Unterbau besitzt, und insbesondere das Muster Front-Controller (statt Transaction-Script) umsetzt, war zur damaligen Zeit keineswegs selbstverständlich.

Clients ermöglicht.
STANLY_ACOS 1 wird derzeit noch produktiv genutzt und gewartet. Allerdings erfährt es nur noch von Zeit zu Zeit kleine Verbesserungen, denn über die Jahre haben sich trotz anfangs guter Architektur viele historische Altlasten angesammelt,[19] sodass der Aufwand für größere Verbesserungen an diesem System in keinem gesunden Verhältnis mehr zu deren Nutzen steht.

STANLY_ACOS FABEC ist ein Prototyp, mit dem die DFS im Frühjahr 2011 an einem von FABEC[20] organisierten Feldversuch (Trial) teilnahm. Ziel des Feldversuchs war, die Möglichkeiten zum europäischen Austausch der Luftraumbuchungs-Daten zu evaluieren, um in Zukunft die Luftraumnutzung über Ländergrenzen hinweg zu optimieren, vor allem in den Lufträumen nahe der Grenzen. STANLY_ACOS FABEC umfasst nur einen Bruchteil der Funktionalitäten von STANLY_ACOS 1, wurde aber mit den Fähigkeiten zum Import und zur gleichzeitigen Darstellung der Luftraumbuchungen auf europäischer Ebene ausgestattet.
Gleichzeitig diente STANLY_ACOS FABEC zum Austesten neuer GUI-Konzepte, zum Austesten einer neuen Gesamt-Architektur und zum Austesten neuer Frameworks auf Server- und Clientseite. Insbesondere ist STANLY_ACOS FABEC eine echte Rich Internet Application (RIA), die im Browser als eigenständige Client-Applikation läuft. STANLY_ACOS 1 hingegen befindet sich auf dem Mittelweg zwischen einer klassischen Webapplikation und einer Rich Internet Application.

STANLY_ACOS 2 wurde nach dem Vorbild von STANLY_ACOS FABEC entwickelt, als der Feldversuch vorüber war. Die Architektur wurde jedoch nicht 1:1 übernommen, sondern es flossen bei der Gelegenheit einige Verbesserungen ein, basierend auf den Erfahrungen während der Entwicklung von STANLY_ACOS FABEC.[21] STANLY_ACOS 2 ist ebenfalls eine Rich Internet Application.
Da STANLY_ACOS 1 nur noch kleine Verbesserungen erfährt, werden neue Funktionalitäten direkt in STANLY_ACOS 2 statt STANLY_ACOS 1 implementiert. Beide Systeme werden parallel betrieben, was durch eine Synchronisation (d.h. einen ständigen Import) der Luftraumbuchungsdaten von STANLY_ACOS 1 nach STANLY_ACOS 2 ermöglicht wird. Sobald STANLY_ACOS 2 alle alten Funktionalitäten[22] von STANLY_ACOS 1 nachgerüstet hat, kann ein endgültiger Umstieg stattfinden.

[19] STANLY_ACOS 1 wurde über die Zeit in verschiedenste Richtungen erweitert, um den neuen Bedürfnissen gerecht zu werden. Unter anderem werden keine reinen HTML-Seiten mehr erzeugt, sondern diese enthalten einen großen Anteil an JavaScript-Code, der leider keiner einheitlichen Architektur folgt. Auch die Geschäftsregeln (Business Rules) wurden ständig erweitert, und dabei nicht immer sämtliche Konsequenzen in Bezug auf die Kombination mit den bereits vorhandenen Regeln bedacht.

[20] Ein FAB (Functional Airspace Block) ist ein Zusammenschluss von Flugsicherungen verschiedener Länder. FABs werden errichtet, um den europäischen Luftraum zu „defragmentieren“. [EUR12b] In Europa gibt es neun FABs, von denen FABEC (FAB Europe Central) einer der verkehrsreichsten ist. FABEC besteht aus Deutschland, Frankreich, Belgien, den Niederlanden, Luxemburg und der Schweiz.

[21] Eine genaue Beschreibung dieser Architektur wird in Abschnitt 3.2 vorgenommen.

[22] zumindest die Funktionalitäten, die noch tatsächlich benutzt werden

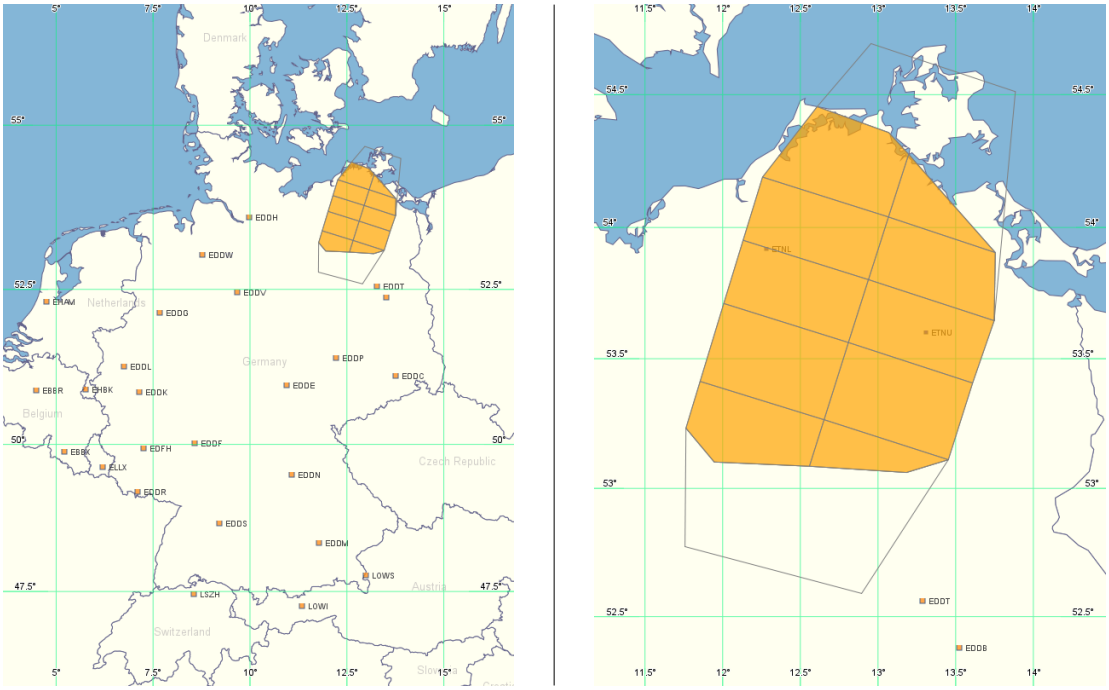


Abbildung 5.: Die MVPA „NEAST“ im Nordosten Deutschlands
(Karten aus STANLY_ACOS 1.3b)

Wenn in dieser Arbeit von STANLY_ACOS die Rede ist, so ist stets STANLY_ACOS 2 gemeint.

Seit 2011 gibt es eine Software-Lizenz, die für alle entwickelten STANLY_ACOS-Varianten gilt (siehe Anhang B). Es handelt sich um eine Abwandlung der 3-Klausel-BSD-Lizenz[23], die um eine eigene DFS-spezifische Zusatzklausel erweitert wurde. Durch diese Zusatzklausel ist STANLY_ACOS keine Freie Software und kein Open Source. Es kann insbesondere nicht ohne Weiteres auf Software-Komponenten aufbauen, die unter GPL oder AGPL lizensiert sind. Dennoch ist diese Lizensierung ein großer Schritt in Richtung Offenheit, vor allem gegenüber den Flugsicherungen anderer Länder.

1.5. Fragestellungen

Diese Diplomarbeit baut auf der Vorarbeit von Nils Hartwig auf, der eine Erhebung der existierenden BNL-Arbeitsabläufe vornahm und daraus eine umfangreiche Liste von funktionalen Anforderungen an eine BNL-Software ableitete. [Har12]

In dieser Diplomarbeit soll nun von Seiten der Software-Architektur untersucht werden, wie solch eine BNL-Software aufgebaut sein könnte, und inwieweit diese sinnvoll in das bestehende STANLY_ACOS integriert werden kann oder auch nicht. Hierzu soll

[23] auch als *New BSD License* bekannt

zunächst eine zweite, eigene Anforderungserhebung durchgeführt werden, die die erste verfeinert und ergänzt. Der Schwerpunkt soll dabei vor allem auf der Erfassung nichtfunktionaler Anforderungen liegen. Auf dieser Grundlage sollen mögliche Architekturen entworfen und gegenübergestellt werden, um dann zu evaluieren, welche Architektur am geeignetsten ist. Die gewählte Architektur soll in Form eines Prototypen umgesetzt werden, um zu überprüfen, inwieweit diese tatsächlich tragfähig ist. Idealerweise sollte der Prototyp zudem präsentierfähig gegenüber Nutzern sein, um so eine Diskussions-Grundlage für zukünftige Entwicklungen zu schaffen.

1.6. Ergebnisse

Die zweite Anforderungserhebung erwies sich als äußerst fruchtbar. Es konnten Definitionslücken und Unklarheiten in insgesamt 8 der 46 Anforderungen beseitigt werden. Zudem sind 9 Anforderungen neu hinzugekommen, davon 3 funktionale und 6 nichtfunktionale.

Auf Basis der so aufbereiteten Anforderungslage konnten zahlreiche Architekturen entworfen werden. Hierbei kam eine sehr offene, systematische Methode zum Einsatz, um ein möglichst breites Spektrum von Lösungs-Architekturen zu erforschen. Dabei ergaben sich 4 sinnvolle Kommunikationsstrukturen, für die es insgesamt 78 funktionierende Kombinationen von Komponenten gab, die allesamt die höchstpriorisierten Anforderungen erfüllen. Die Anforderungen waren ebenfalls ausreichend, um eine Evaluation all dieser 78 Architekturen durchzuführen und die am besten geeignete für den Prototyp auszuwählen.

Der Prototyp konnte in der gegebenen Zeit fertiggestellt werden. Die erhofften Synergie-Effekte durch Realisierung als Modul in STANLY_ACOS traten überwiegend ein. Dennoch gab es Funktionalitäten, wie die automatische Aktualisierung und die Benutzerführung, die zwar in STANLY_ACOS vorhanden waren, aber für den Prototypen nicht direkt wiederverwendet werden konnten. Unerwartete Schwierigkeiten bereitete zudem das von STANLY_ACOS clientseitig verwendeten ExtJS-Framework, das es äußerst schwierig macht, eine robuste Benutzerführung zu etablieren, die alle Randfälle abdeckt. Letztendlich konnten jedoch alle im Prototyp entdeckten Probleme mit akzeptablem Aufwand gelöst werden.

Der Prototyp bestätigt damit die Tragfähigkeit der Architektur. Das sekundäre Ziel des Prototypen als präsentierfähige Diskussions-Grundlage wurde ebenfalls erreicht.

1.7. Aufbau der Arbeit

Die Struktur der Arbeit orientiert sich direkt an der Aufgabenstellung.

Kapitel 1 – Einführung bietet einen allgemeinverständlichen Einstieg in die Flugsicherung und speziell in das Thema *Besondere Nutzung Luftraum*. Danach wird die präzise Aufgabenstellung formuliert und die verwendeten Notationen erklärt.

Kapitel 2 – Anforderungs-Spezifikation fasst alle für diese Arbeit relevanten Anforderungen zusammen. Dabei handelt es sich um eine konsolidierte Fassung aus den ursprünglichen Anforderungen und der eigenen Erhebung. Der Eigenanteil ist jeweils klar gekennzeichnet.

Kapitel 3 – Lösungs-Architekturen behandelt die systematisch erzeugten Architektur-Entwürfe für die BNL-Software, die auf 78 sinnvolle Kandidaten reduziert werden. Zudem wird die Architektur von STANLY_ACOS erklärt.

Kapitel 4 – Evaluation beinhaltet die entsprechende Evaluation der Architekturen. Hier fällt die Entscheidung für die Architektur des Prototypen.

Kapitel 5 – Prototypische Umsetzung beschreibt den Aufbau und die Funktionsweise des Prototypen. Dabei werden die Einschränkungen des Prototypen klargestellt und es wird auf die Herausforderungen während der Entwicklung eingegangen.

Kapitel 6 – Schlussfolgerungen und Ausblick fasst die gewonnenen Erkenntnisse zusammen, zieht entsprechende Schlussfolgerungen und gibt einen Ausblick auf mögliche zukünftige Entwicklungen. Es werden weiterführende wissenschaftliche Fragen gestellt, die in dieser Arbeit nicht untersucht werden konnten.

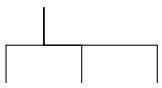
Die Anhänge enthalten zusätzliches Material, etwa eine Beschreibung der Systemumgebung des Prototypen. Auch wird das Hilfsprogramm aufgeführt, mit dem einige Tabellen dieser Arbeit generiert wurden, vor allem für die Evaluation.

1.8. Notationen

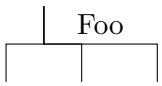
In dieser Arbeit werden an mehreren Stellen Diagramme verwendet, um die im Text beschriebenen Sachverhalte zu veranschaulichen, etwa Abbildung 6 für die Netzwerkstruktur oder Abbildung 10 für die Architektur BS-IS/*.

Diese Diagramme entsprechen keinem speziellen Standard, da es bislang keine Standard-Notation für Software-Architekturen gibt und zudem fraglich ist, ob es so etwas jemals geben wird. Auch in der Fachliteratur werden Software-Architekturen stets in einem pragmatischen Stil mit möglichst wenigen, schlichten Elementen ausgedrückt. Zwar gibt es Standards wie UML, doch diese befassen sich nicht mit der Architektur, sondern mit Elementen der Spezifikations-Phase (z.B. Anwendungsfälle) oder der Design-Phase (z.B. Klassendiagramme).

Die Diagramme in dieser Arbeit wurden zunächst ad hoc erstellt, wobei wiederkehrende Elemente stets in allen Diagrammen gleichartig notiert wurden. Es folgt eine Legende der auf diese Weise entstandenen Notation:



Kommunikationsnetzwerk (Computer- oder Telefon/Fax-Netzwerk) bzw. innerhalb einer Software stattfindene Kommunikation



Kommunikationsnetzwerk „Foo“

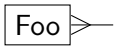
Kommunikationsweg in einem Netzwerk bzw. innerhalb einer Software

bedingter Kommunikationsweg (z.B. nur für Debugging-Zwecke)

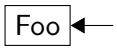
Komponente „Foo“

hervorgehobene Komponente „Foo“

unwichtige Komponente „Foo“ (zum Beispiel, um die übrigen Komponenten in einen größeren Zusammenhang zu stellen)

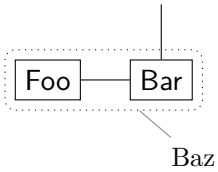


viele Komponenten „Foo“ sind mit dem Kommunikationsnetzwerk verbunden



Komponente „Foo“ kann über den Kommunikationsweg kontaktiert werden (z.B. eine Serverkomponente, die TCP/IP-Verbindungen aus diesem Netzwerk annimmt)

Komponente „Foo“ kann über den linken, aber nicht über den rechten Kommunikationsweg kontaktiert werden



Komponenten „Foo“ und „Bar“ bilden gemeinsam die größere Komponente „Baz“

menschlicher Kommunikationsteilnehmer

maschineller Kommunikationsteilnehmer (Arbeitsplatzrechner, Server, …)

spezialisierter maschineller Kommunikationsteilnehmer (Telefon, Fax)

problematische Stelle im Konzept

2. Anforderungs-Spezifikation

In diesem Kapitel werden sämtliche Anforderungen spezifiziert, die an das BNL-Modul im Rahmen dieser Arbeit gestellt werden. Sie bilden die Grundlage für die Architektur-Entwürfe, und sind zugleich Maßstab für die Auswahl der Architektur des Prototypen. Im Prototyp selbst wird nur ein Teil dieser Anforderungen implementiert, der jedoch ausreicht, um die Tragfähigkeit der Architektur zu demonstrieren.

Als Basis für die Anforderungen dient das Dokument *BNL Funktionalität* [Har12] von Nils Hartwig, der bereits eine sehr detaillierte Erhebung der funktionalen Anforderungen durchgeführt hat. Diese werden um nichtfunktionale Anforderungen ergänzt, die auf einer eigenen Anforderungserhebung beruhen. Aus dieser zweiten Erhebung ergaben sich auch einige Änderungen und Ergänzungen an den ursprünglichen funktionalen Anforderungen.

Um Verwechselungen zu vermeiden, sind die Anforderungen in dieser Arbeit bewusst nach einem anderen Schema nummeriert als im Dokument *BNL Funktionalität*. Zu jeder Anforderung ist vermerkt, aus welcher ursprünglichen Anforderung sie hervorgegangen ist (falls sie nicht komplett auf eigenen Recherchen beruht), und inwieweit sie gegebenenfalls von den ursprünglichen Anforderungen abweicht.

Die Anforderungen werden in funktionale und nichtfunktionale Anforderungen unterteilt. Darüber hinaus sind sie in folgende Prioritätsstufen eingeteilt:

notwendig (Anforderungen 1 – 44)
: Diese Anforderungen müssen unbedingt umgesetzt werden. Sie werden im Dokument *BNL Funktionalität* auch *mandatory* genannt.

wünschenswert (Anforderungen 45 – 49)
: Unter den nicht-notwendigen Anforderungen genießen diese die höchste Priorität. Sie werden im Dokument *BNL Funktionalität* auch *preferred* genannt.

optional (Anforderungen 50 – 55)
: Diese Anforderungen haben eine geringere Priorität. Sie werden im Dokument *BNL Funktionalität* auch als *optional* oder *future development* bezeichnet.

2.1. Notwendig

Anforderung 1 (Dateneingabe – *funktional, notwendig*)
Es soll ein neues Vorhaben in das BNL-Modul eingegeben werden können. Ein Vorhaben besteht aus:

Feldname	Deutsche Bezeichnung
Type	Art des Vorhabens
Unique ID	Unique ID
Begin	Startzeit, Datum
End	Endzeit, Datum
Map/ID name	Segelflugsektorname, Kunstflugboxname, Fallschirmsprungzonenname, Vorhabenkartenname
Geometry	Flächenkoordinaten
Min height	min. Höhe
Max height	max. Höhe
Separation buffer	Staffelungsbereich
Contact	Kontaktname
Company / Aerodrome	Kontaktfirma/Kontaktflugplatz
Address	Kontaktanschrift
Fax	Kontaktfax
Email	Kontaktemail
Phone	Kontakttelefon
Always available	Kontakttelefonnummer für ständige Erreichbarkeit
Departure	Startplatz
Destination	Landeplatz
Aircraft	LFZ-Kennzeichen
PJE type	Art des Fallschirmsprungvorhabens
Drop offs	Anzahl der Absetzvorgänge
Parachutes	Anzahl der Springer
Reference	Referenzangabe
Sector	Kontrollsektor
Comment	Bemerkungen/Beschränkungen

Dabei gibt es folgende Typen (Arten) von Vorhaben:[1]

Kürzel	Typ	Deutsche Bezeichnung
Acro	Aerobatic	Kunstflug
PJE-MIL	Parachute Jumping Exercise MIL	Fallschirmsprungvorhaben SPZ
PJE-NOTAM	Parachute Jumping Exercise NOTAM	Fallschirmsprungvorhaben NOTAM
PJE-SPZ	Parachute Jumping Exercise SPZ	Fallschirmsprungvorhaben MIL
Gas	Gas Blowout	Gasausblasung
Bomb	Bomb Disposal	Bombenentschärfung
Airshow	Airshow	Luftfahrtveranstaltung
Gli	Gliding	Segelflug
Scan	Scanning Flights	Fotoflüge
Others	Others	Sonstige

[1] Die in dieser Tabelle aufgeführten Kürzel sind keine offiziellen Kürzel, sondern frei erfunden!

Je nach Typ wird nur ein Teil der Eingabefelder benötigt:

	Acro	PJE-*	Gas	Bomb	Airshow	Gli	Scan	Others
Map/ID name	✓	✓	—	—	—	✓	—	—
Separation buffer	✓	✓	✓	✓	✓	—	—	✓
Departure	✓	✓	—	—	✓	—	✓	✓
Destination	✓	✓	—	—	✓	—	✓	✓
Aircraft	✓	✓	—	—	—	—	✓	✓
PJE type	—	✓	—	—	—	—	—	—
Drop offs	—	✓	—	—	—	—	—	—
Parachutes	—	✓	—	—	—	—	—	—
Reference	—	✓	—	—	—	—	—	—
Sector	—	—	—	—	—	✓	—	—
Alle anderen Felder	✓	✓	✓	✓	✓	✓	✓	✓

Die Höhenangaben *Min height* und *Max height* sollen in folgenden Einheiten spezifiziert werden können:

Kürzel	Einheit	Umrechnung
m	Meter	—
ft	Fuß	1ft = 0,3048m [Cen06]
FL	Flight Level	entspricht 100ft unter Normaldruck

Die Maßeinheiten für *Separation buffer* sind in Anforderung 45 spezifiziert.

Im Gegensatz zu der ursprünglichen Anforderung sind einige englische Feldnamen korrigiert, und zudem bei Bedarf verkürzt, sodass sie zugleich sinnvolle Bezeichner für die Eingabefelder sind. Die Vorhabens-Typen sind um Kürzel ergänzt (für tabellarische Darstellungen). Die Felder *Segelflugsektorname*, *Kunstflugboxname*, *Fallschirmsprungzonenname* und *Vorhabenkartenname* sind zu einem gemeinsamen Feld zusammengefasst, da sich die konkrete Auswahlliste dieser Grunddaten-Lufträume aus dem Kontext der ausgewählten Art des Vorhabens ergibt. Die Maßeinheiten für Höhenangaben wurden um FL ergänzt. Die Einheit NM wurde für Höhenangaben gestrichen, da diese ungefähr einer Bogenminute auf der Erdoberfläche entspricht und daher nur für horizontale Längenangaben verwendet wird. Weiterhin ist die Möglichkeit vorgesehen, für Start- und Endzeitpunkt unterschiedliche Datumsangaben zu machen (siehe auch Anforderung 4).

(basiert auf eigenen Recherchen und BNL EIN 2 M: Dateneingabe in STANLY_ACOS 2.0 [Har12, S. 14])

Anforderung 2 (Eindeutige Kennzahl – *funktional, notwendig*)
Jedes Vorhaben muss eine eindeutige Kennzahl (Unique ID) erhalten. Sie hat das Format:

`BNL_`*`NN_ZZZZ-JJ`*

Dabei ist:

NN die Niederlassung (GG, MM oder WW),

ZZZZ eine fortlaufende Zahl, die über alle Niederlassungen hinweg gezählt wird, und **JJ** das Jahr.

Besteht ein Vorhaben aus mehreren Teilvorhaben, etwa bei mehrtägigen Vorhaben oder bei Luftfahrtveranstaltungen mit mehreren Teilen (siehe Anforderung 5), so erhalten diese Teilvorhaben jeweils eine eigene eindeutige Kennzahl:

`BNL_`*`NN_ZZZZ-JJ-X`*

Der Anhang *X* ist hierbei ein fortlaufender Kleinbuchstabe.
(basiert auf BNL EIN 3 M: Unique ID [Har12, S. 19])

Anforderung 3 (Zeiteingabe – *funktional, notwendig*)
Zeitangaben sind minutengenau und in lokaler Zeitzone (CET/CEST). Zeiten sollen ohne Doppelpunkt eingegeben werden können: „`1540`“ → „`15:40`“.
(basiert auf BNL EIN 4 M: Zeiteingabe [Har12, S. 19])

Anforderung 4 (Zeitbereichseingabe – *funktional, notwendig*)
Ein Vorhaben kann an einem oder an mehreren Tagen stattfinden. Die Tage sollen via Datepicker eingegeben werden können. Dabei sind folgende Eingaben möglich (häufigste zur seltensten):

1. der heutige Tag
2. ein anderer Tag
3. mehrere aufeinanderfolgende Tage
4. mehrere beliebige Zeitbereiche

Es können für jeden Tag andere Uhrzeiten gesetzt werden. Jedoch soll es auch möglich sein, alle Tage mit einem gemeinsamen Uhrzeit-Bereich vorzubelegen.

Zudem ist es möglich, dass ein Vorhaben zu mehreren Zeiten an einem Tag stattfindet, etwa 10–11 Uhr und 17–18 Uhr.

Der Zeitbereich ist nicht an Tagesgrenzen gebunden. Das heißt, der Nutzer sollte ein Vorhaben von 23–2 Uhr eingeben können und nicht dazu gezwungen sein, es in zwei Vorhaben 23–0 Uhr und 0–2 Uhr aufzuspalten.

Es ist sogar möglich, dass der Zeitbereich mehr als einen vollen Tag umfasst, zum Beispiel beginnend am Dienstag 20 Uhr, den gesamten Mittwoch hindurch, bis zum Donnerstag 10 Uhr.

Im Gegensatz zu der ursprünglichen Anforderung wurden noch weitere Möglichkeiten ergänzt, die allesamt auftreten können. Diese zeigen, dass ein Vorhaben in mehreren Zeitbereichen stattfinden kann kann, deren Start- und Endzeitpunkt sich an unterschiedlichen Tagen befinden können. Die in den ursprünglichen Anforderungen angedeutete Auftrennung in Datum und Uhrzeit-Bereich erwies sich als zu unflexibel und entsprach nicht dem tatsächlichen Bedarf. Diese Erkenntnis wurde auch in Anforderung 1 berücksichtigt.

(basiert auf eigenen Recherchen und BNL EIN 5 M: Datumseingabe [Har12, S. 20])

Anforderung 5 (Eingabe von Luftfahrtveranstaltungen – *funktional, notwendig*)
Es soll möglich sein, mehrteilige Vorhaben einzugeben. Dies betrifft alle Vorhaben vom Typ *Airshow* (Luftfahrtveranstaltung). Die Teilvorhaben werden wie in Anforderung 2 beschrieben mit einem Kleinbuchstaben als Anhang in der *Unique ID* gekennzeichnet. Alle Teilvorhaben basieren auf der selben Geometrie (siehe Anforderung 6), können aber unterschiedliche Zeitbereiche und unterschiedliche Höhen belegen.

Eine Luftfahrtveranstaltung findet in der Regel an mehreren Tagen statt. Für jeden Tag werden die meisten Aktivitäten in einem Teilvorhaben vom Typ *Display* zusammengefasst. Dieses umfasst den gesamten Zeitraum der Veranstaltung, belegt aber nur eine geringe Höhe, sodass der normale Flugverkehr kaum beeinträchtigt wird.

Einige dieser Aktivitäten, Kunstflug (Typ *Acro*) und Fallschirmsprünge (Typ *PJE-**), erfordern jedoch die kurzfristige Sperrung höherer Lufträume. Dies geschieht durch zusätzliche Teilvorhaben mit geringeren Zeitbereichen und größeren Höhenbereichen. Nur zu diesen Zeiten wird der normale Flugverkehr entsprechend beeinträchtigt.

Eine Luftfahrtveranstaltung könnte beispielsweise so aussehen:

Typ		Zeitbereich	Höhe	Unique ID
Display		2014-01-01 11:00–18:00	0–200m	BNL_GG_0001-11-a
	Acro	11:00–12:00	0–600m	BNL_GG_0001-11-b
	Acro	13:00–13:30	0–600m	BNL_GG_0001-11-c
	PJE	17:30–18:00	0–2500ft	BNL_GG_0001-11-d
Display		2014-01-02 12:00–15:00	0–200m	BNL_GG_0001-11-e
	PJE	12:00–12:30	0–2500ft	BNL_GG_0001-11-f
	Acro	14:00–14:30	0–600m	BNL_GG_0001-11-g
Display		2014-01-04 12:00–13:00	0–200m	BNL_GG_0001-11-h

In der Karte zählen die Teilvorhaben sowohl zu ihrem eigenen Typ, als auch zum Typ des umschließenden Gesamtvorhabens. Das heißt, das Teilvorhaben *BNL_GG_0001-11-d* in obigem Beispiel soll sowohl im Karten-Layer für Fallschirmsprünge als auch im Karten-Layer für Luftfahrtveranstaltungen sichtbar sein.

(basiert auf BNL EIN 6 M: Eingabe von Luftfahrtveranstaltungen [Har12, S. 21])

Anforderung 6 (Mögliche Geometrien – *funktional, notwendig*)
Folgende Arten von Geometrien sollen eingegeben werden:

1. Kreise
2. Polygone
3. Streckenzüge

Dabei werden die Streckenzüge nur importiert (siehe Anforderung 8), und nicht von Hand eingegeben.

(basiert auf BNL EIN 8 M: Abbildungsformen der Koordinaten [Har12, S. 23])

Anforderung 7 (Syntax der Koordinaten – *funktional, notwendig*)
Die Eingabe der Geometrien aus Anforderung 6 soll stets in Koordinatenform möglich sein. Als Koordinatensystem kommt *EPSG:4326* zum Einsatz. Das heißt, die Koordina-

ten sind Längen- und Breitengrad in Bezug auf den in *WGS 84* definierten Referenzellipsoiden. [OGP14c] Die Syntax der Koordinaten ist:

$$GG°MM'SS"\text{N} \quad GGG°MM'SS"\text{E}$$

wobei die Koordinaten in Grad, Minuten und Sekunden angegeben werden. Die Symbole sollen automatisch ergänzt werden, sodass auch die Eingabe des reinen Zahlencodes ausreicht, zum Beispiel:

$$0080051 \rightarrow 008°00'51"\text{E}$$

(basiert auf BNL EIN 7 M: Syntax der Koordinaten [Har12, S. 23])

Anforderung 8 (Koordinaten-Eingabe – *funktional, notwendig*)
Die Koordinaten der Geometrien aus Anforderung 6 sollen auf folgende Arten eingegeben und editiert werden:

1. Manuelle Eingabe der Koordinaten
2. Markieren auf der Karte
3. Import aus einer Datei

Für den Import sollen mindestens das OVL-Format[2] und ein CSV-Format[3] unterstützt werden.

Im Gegensatz zu der ursprünglichen Anforderung wird die Unterstützung von CSV anstelle von XLS gefordert.[4] Letzteres wird stattdessen in einer separaten Anforderung 51 von geringerer Priorität genannt.

(basiert auf eigenen Recherchen und BNL EIN 9 M: Eingabemöglichkeiten der Koordinaten [Har12, S. 24])

Anforderung 9 (Polygon-Eingabe – *funktional, notwendig*)
Die Koordinaten von Polygonen sollen zugleich sowohl auf der Karte als auch als Text editierbar sein (siehe Anforderung 8). Zudem sollen Punkte auch nachträglich noch hinzugefügt und entfernt werden können.

(basiert auf BNL EIN 10 M: Eingabe der Polygonkoordinaten [Har12, S. 25])

Anforderung 10 (Kreis-Eingabe – *funktional, notwendig*)
Ein Kreis soll in Form von Mittelpunkt (entsprechend Anforderung 8) und Radius eingegeben werden. Für den Radius sollen die gleichen Maßeinheiten wie für den Staffelungsbereich in Anforderung 45 unterstützt werden.

(basiert auf BNL EIN 11 M: Eingabe der Kreiskoordinaten [Har12, S. 25])

[2] Gemeint ist hier die ASCII-Variante von OVL. Es handelt sich um ein Plaintext-Datenformat im Stil von Ini-Dateien. [Cac10]

[3] CSV steht für *Comma-Separated Values* und ist ein Sonderfall von DSV (*Delimiter-Separated Values*). [Ray03, S. 137 ff.]

[4] Die Intention dieser Anforderung war es, Antragstellern die Koordinaten-Übergabe in einem einfach zu erzeugenden Standardformat zu ermöglichen. Dieses Ziel kann jedoch mit dem XLS-Format von Microsoft Excel nicht erreicht werden, da es sehr komplex und kein offener Standard ist. Ein CSV-Format hingegen erfüllt diesen Zweck sehr gut.

Anforderung 11 (Auflösung von Adressen – *funktional, notwendig*)
Ortsangaben in Form von Adressen oder Straßenkreuzungen sollen in Koordinaten umgewandelt werden. Hierzu könnte das Nominatim von OpenStreetMap verwendet werden. [Ope10c]

(basiert auf BNL EIN 12 M: Koordinatentransformation I [Har12, S. 26])

Anforderung 12 (Gauß-Krüger-Koordinaten – *funktional, notwendig*)
Diese Anforderung wurde aus [Har12] nicht übertragen, da sie für den Inhalt dieser Diplomarbeit keine Relevanz hat.

(basiert auf BNL EIN 13 M: Koordinatentransformation II [Har12, S. 27])

Anforderung 13 (Speichern der Daten – *funktional, notwendig*)
Diese Anforderung wurde aus [Har12] nicht übertragen, da sie für den Inhalt dieser Diplomarbeit keine Relevanz hat.

(basiert auf BNL EIN 17 M: Speichern der Daten [Har12, S. 28])

Anforderung 14 (Direkteingabe durch Antragsteller – *funktional, notwendig*)
Ein Vorhaben soll direkt durch den Antragsteller eingegeben werden können, sodass die Daten auf Seiten der DFS nicht nochmals eingegeben werden müssen. Das verringert die Gefahr von Fehleingaben und Missverständnissen. Zudem können akute, kritische Vorhaben schneller bearbeitet werden.

Die Eingabemaske soll den Anforderungen 1, 2, 3, 4 und 5 genügen. Weiterhin soll sie ein direktes Feedback gemäß Anforderung 47 bieten, um fehlerhaften Koordinateneingaben vorzubeugen.

Im Gegensatz zu der ursprünglichen Anforderung erhält diese Anforderung eine sehr hohe Priorität (*notwendig* statt *future development*). Zudem ist sie allgemeiner formuliert (*Direkteingabe* statt *Webinterface*), um keine technologischen Erwägungen vorweg zu nehmen.

(basiert auf eigenen Recherchen und BNL EIN 18 F: Webinterface [Har12, S. 29])

Anforderung 15 (Inhalt der BNL-Tabelle – *funktional, notwendig*)
Diese Anforderung wurde aus [Har12] nicht übertragen, da sie für den Inhalt dieser Diplomarbeit keine Relevanz hat.

(basiert auf BNL TAB 1 M: Input durch Eingabemaske [Har12, S. 30])

Anforderung 16 (Filtern und Sortieren in BNL-Tabelle – *funktional, notwendig*)
Diese Anforderung wurde aus [Har12] nicht übertragen, da sie für den Inhalt dieser Diplomarbeit keine Relevanz hat.

(basiert auf BNL TAB 2 M: Filtern und Sortieren im Bookintable [Har12, S. 30])

Anforderung 17 (Suchen in BNL-Tabelle – *funktional, notwendig*)
Diese Anforderung wurde aus [Har12] nicht übertragen, da sie für den Inhalt dieser Diplomarbeit keine Relevanz hat.

(basiert auf BNL TAB 3 M: Suchen im Bookingtable [Har12, S. 31])

Anforderung 18 (Selektieren in BNL-Tabelle und Karte – *funktional, notwendig*)
Wird ein Vorhaben in der Liste ausgewählt, so soll dessen Geometrie in der Karte hervorgehoben werden. Dies soll in einem separaten Layer der Karte geschehen, sodass das ausgewählte Vorhaben auch dann sichtbar ist, wenn der Nutzer eigentlich die Vorhaben in der Karte ausgeblendet hat.

Weiterhin soll ein Vorhaben durch Klick auf dessen Geometrie in der Karte ausgewählt werden können.

(basiert auf BNL TAB 4 M: Markieren im Bookingtable [Har12, S. 33])

Anforderung 19 (Anzeige auf der Karte – *funktional, notwendig*)
Diese Anforderung wurde aus [Har12] nicht übertragen, da sie für den Inhalt dieser Diplomarbeit keine Relevanz hat.

(basiert auf BNL TAB 5 M: Anzeigen auf der Map [Har12, S. 33])

Anforderung 20 (Löschen in BNL-Tabelle – *funktional, notwendig*)
Diese Anforderung wurde aus [Har12] nicht übertragen, da sie für den Inhalt dieser Diplomarbeit keine Relevanz hat.

(basiert auf BNL TAB 6 M: Löschen im Bookingtable [Har12, S. 34])

Anforderung 21 (Farben in BNL-Tabelle – *funktional, notwendig*)
Diese Anforderung wurde aus [Har12] nicht übertragen, da sie für den Inhalt dieser Diplomarbeit keine Relevanz hat.

(basiert auf BNL TAB 7 M: Farben setzen im Bookingtable [Har12, S. 34])

Anforderung 22 (Aktivierungs-Zustände in BNL-Tabelle – *funktional, notwendig*)
Diese Anforderung wurde aus [Har12] nicht übertragen, da sie für den Inhalt dieser Diplomarbeit keine Relevanz hat.

(basiert auf BNL TAB 8 M: Aktivieren/Deaktivieren im Bookingtable [Har12, S. 35])

Anforderung 23 (Nachträgliche Änderung – *funktional, notwendig*)
Diese Anforderung wurde aus [Har12] nicht übertragen, da sie für den Inhalt dieser Diplomarbeit keine Relevanz hat.

(basiert auf BNL TAB 9 M: Nachträgliche Änderung [Har12, S. 36])

Anforderung 24 (Layer in der Karte – *funktional, notwendig*)
Diese Anforderung wurde aus [Har12] nicht übertragen, da sie für den Inhalt dieser Diplomarbeit keine Relevanz hat.

(basiert auf eigenen Recherchen und BNL MAP 1 M: BNL Layer [Har12, S. 37])

Anforderung 25 (Fotoflug-Layer – *funktional, notwendig*)
Diese Anforderung wurde aus [Har12] nicht übertragen, da sie für den Inhalt dieser Diplomarbeit keine Relevanz hat.

(basiert auf BNL MAP 2 M: Layer der Fotoflüge [Har12, S. 38])

Anforderung 26 (RadGIS-Layer – *funktional, notwendig*)
Diese Anforderung wurde aus [Har12] nicht übertragen, da sie für den Inhalt dieser Diplomarbeit keine Relevanz hat.

(basiert auf BNL MAP 3 M: RadGIS Layer [Har12, S. 38])

Anforderung 27 (Optische Erscheinung auf der Karte – *funktional, notwendig*)
Diese Anforderung wurde aus [Har12] nicht übertragen, da sie für den Inhalt dieser Diplomarbeit keine Relevanz hat.
(basiert auf BNL MAP 4 M: Optische Unterscheidung auf der Map [Har12, S. 39])

Anforderung 28 (Import der Grunddaten – *funktional, notwendig*)
Diese Anforderung wurde aus [Har12] nicht übertragen, da sie für den Inhalt dieser Diplomarbeit keine Relevanz hat.
(basiert auf BNL QDB 1 M: Daten für die Grunddatenbanken importieren [Har12, S. 40])

Anforderung 29 (RadGIS-Import – *funktional, notwendig*)
Diese Anforderung wurde aus [Har12] nicht übertragen, da sie für den Inhalt dieser Diplomarbeit keine Relevanz hat.
(basiert auf BNL QDB 2 M: Daten in die RadGIS DB importieren [Har12, S. 41])

Anforderung 30 (NOTAM-Import – *funktional, notwendig*)
Diese Anforderung wurde aus [Har12] nicht übertragen, da sie für den Inhalt dieser Diplomarbeit keine Relevanz hat.
(basiert auf BNL QDB 3 M: Daten aus den NOTAMs in die NOTAM DB importieren [Har12, S. 42])

Anforderung 31 (AIP-ENR-5.5-Import – *funktional, notwendig*)
Diese Anforderung wurde aus [Har12] nicht übertragen, da sie für den Inhalt dieser Diplomarbeit keine Relevanz hat.
(basiert auf eigenen Recherchen und BNL QDB 4 M: Daten aus der AIP in die AIP ENR 5.5 DB importieren [Har12, S. 42])

Anforderung 32 (RadGIS-Import für Kunstflug – *funktional, notwendig*)
Diese Anforderung wurde aus [Har12] nicht übertragen, da sie für den Inhalt dieser Diplomarbeit keine Relevanz hat.
(basiert auf BNL QDB 5 M: Daten aus den RadGIS Karten in die Kunstflugbox DB importieren [Har12, S. 43])

Anforderung 33 (RadGIS-Import für Segelflug – *funktional, notwendig*)
Diese Anforderung wurde aus [Har12] nicht übertragen, da sie für den Inhalt dieser Diplomarbeit keine Relevanz hat.
(basiert auf BNL QDB 6 M: Daten aus den RadGIS Karten in die Segelflug DB importieren [Har12, S. 43])

Anforderung 34 (BNL-Daten – *funktional, notwendig*)
Diese Anforderung wurde aus [Har12] nicht übertragen, da sie für den Inhalt dieser Diplomarbeit keine Relevanz hat.
(basiert auf BNL DAB 1 M: Booking DB [Har12, S. 44])

Anforderung 35 (Adressbuch-Daten – *funktional, notwendig*)
Diese Anforderung wurde aus [Har12] nicht übertragen, da sie für den Inhalt dieser Diplomarbeit keine Relevanz hat.
(basiert auf BNL DAB 2 M: Adressbuch DB [Har12, S. 44])

Anforderung 36 (Kunstflug- und Segelflug-Daten – *funktional, notwendig*)
Diese Anforderung wurde aus [Har12] nicht übertragen, da sie für den Inhalt dieser Diplomarbeit keine Relevanz hat.
(basiert auf BNL DAB 3 M: Segellfug DB und Kunstflug DB [Har12, S. 45])

Anforderung 37 (AIP-ENR-5.5- und NOTAM-Daten – *funktional, notwendig*)
Diese Anforderung wurde aus [Har12] nicht übertragen, da sie für den Inhalt dieser Diplomarbeit keine Relevanz hat.
(basiert auf BNL DAB 4 M: AIP ENR 5.5 DB und NOTAM DB [Har12, S. 45])

Anforderung 38 (Drucken von BNL-Vorhaben – *funktional, notwendig*)
Diese Anforderung wurde aus [Har12] nicht übertragen, da sie für den Inhalt dieser Diplomarbeit keine Relevanz hat.
(basiert auf BNL PRI 1 M: Drucken von Bookings [Har12, S. 47])

Anforderung 39 (Nutzertypen – *funktional, notwendig*)
Im Rahmen von BNL gibt es folgende Typen von Nutzern, von denen jedoch vorerst noch nicht alle auf das BNL-Modul zugreifen sollen:

Lotse – vorerst kein Zugriff, sondern erhält bei Bedarf Ausdrucke von Supervisor
Supervisor – Zugriff auf volle Funktionalität
Sachbearbeiter – Zugriff auf volle Funktionalität, wie Supervisor
Antragsteller – Zugriff auf alle in Anforderung 14 beschriebenen Funktionalitäten, alle weiteren Vorgänge nur über Sachbearbeiter oder Supervisor
Externe Stelle – vorerst kein Zugriff, sondern wird durch Sachbearbeiter oder Supervisor informiert

(basiert auf eigenen Recherchen)

Anforderung 40 (Automatische Aktualisierung – *funktional, notwendig*)
Wenn ein Benutzer ein Vorhaben ändert, so soll diese Änderung allen anderen Nutzern mit einer Verzögerung von maximal 2 Sekunden ebenfalls zur Verfügung stehen, ohne dass dafür eine extra Benutzer-Interaktion notwendig ist.

Insbesondere soll der Supervisor jederzeit einen Blick auf die aktuelle BNL-Liste werfen können, ohne dafür einen Reload-Button oder Ähnliches betätigen zu müssen.

Diese Anforderung fiel erst während der Implementierungs-Phase auf, siehe auch Abschnitt 5.7. Sie liegt auf der Grenze zwischen funktionalen und nichtfunktionalen Anforderungen. Sie wurde im Rahmen dieser Arbeit als funktionale Anforderung eingestuft, da sie ein essentielles Feature der Software beschreibt.
(basiert auf eigenen Recherchen)

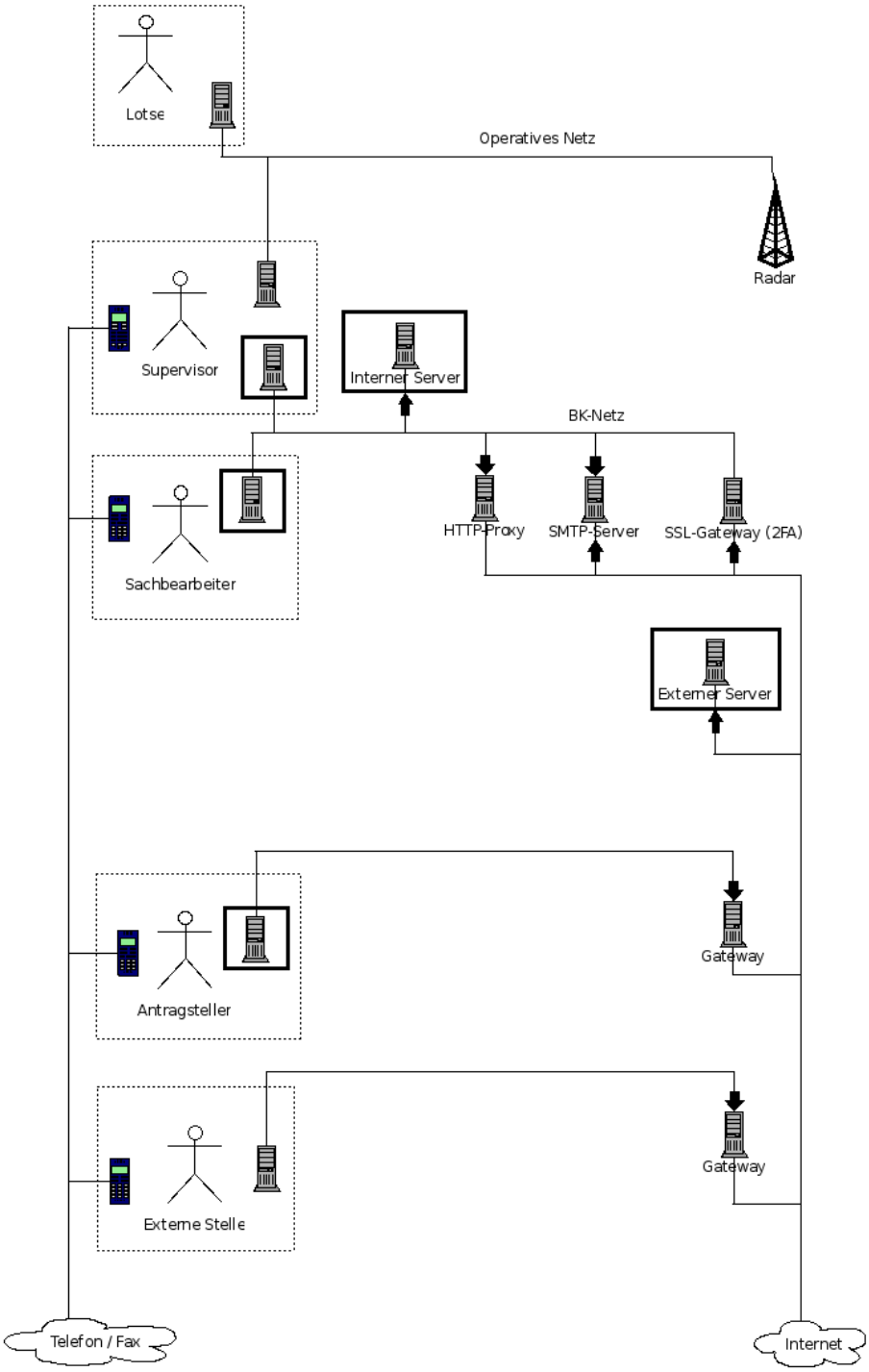


Abbildung 6.: Vereinfachte Darstellung der Netzwerkstruktur, über die die beteiligten Akteure kommunizieren. Die einzig möglichen Ansatzpunkte für das BNL-Modul sind durch eine dicke, durchgezogene, rechteckige Umrandung hervorgehoben. Die Notation ist in Abschnitt 1.8 erklärt.

Anforderung 41 (Netzwerkstruktur – *nichtfunktional, notwendig*)
Das BNL-Modul soll sich in die vorhandene Netzwerkstruktur eingliedern. Diese ist in Abbildung 6 vereinfacht dargestellt. Die vorhandenen sicherheitsrelevanten Kommunikations-Barrieren sollen respektiert werden. Sie sollen nicht durch kreative Techniken wie zum Beispiel Tunneling umgangen werden.

Die Arbeitsplätze der Lotsen hängen ausschließlich am operativen Netz,[5] das physikalisch von allen anderen Netzen getrennt ist. Es darf weder die Software auf den Lotsen-Arbeitsplätzen modifiziert werden, noch darf Software ergänzt werden, noch dürfen zusätzliche Geräte an die Arbeitsplätze gestellt werden, noch dürfen zusätzliche Server in das operative Netz eingebracht werden.[6]

Die Sachbearbeiter haben an ihrem Arbeitsplatz keinen direkten Zugriff auf das operative Netz. Ihre Arbeitsplatz-Rechner sind stattdessen mit dem normalen BK-Netz (Büro-Kommunikation) verbunden.

An den Arbeitsplätzen der Supervisor stehen mehrere separate Geräte, die mit unterschiedlichen Netzwerken verbunden sind. So können die Supervisor wie die Lotsen im operativen Netz arbeiten, aber genauso auch wie die Sachbearbeiter im BK-Netz arbeiten.

Alle Arbeitsplatz-Rechner im BK-Netz sind gleich ausgestattet und haben Zugriff auf die Server im BK-Netz. Auf ihnen läuft ein Browser. Es kann zusätzliche Software installiert werden, wenn diese entsprechende Sicherheitsüberprüfungen besteht. Aus dem BK-Netz heraus haben sie Zugang zum Internet über einen HTTP-Proxy und einen Mailserver. Eine direkte Verbindung ins Internet besteht nicht.

Die Server im BK-Netz[7] sind ebenfalls nicht direkt mit dem Internet verbunden. Jedoch gibt es ein SSL-Gateway, das einzelne HTTP-Server via HTTPS nach außen verfügbar machen kann. Der Zugriff von außen ist personengebunden und durch eine Zwei-Faktor-Authentifizierung abgesichert, die aus einem Hardware-Token und einem Passwort besteht.

Es gibt eine Vielfalt möglicher Antragsteller, daher sollen konservative Annahmen über ihre interne Netzwerkstruktur getroffen werden. Von einem einfachen Subnetzwerk hinter einem Standard-Router bis hin zu einem abgeschotteten Subnetzwerk ähnlich dem BK-Netz der DFS ist alles denkbar. Die Antragsteller, die oft BNL-Anträge stellen, sind prinzipiell auch zur Nutzug einer entsprechenden Client-Software bereit. Ein Browser sowie E-Mail können dort ebenfalls vorausgesetzt werden.

[5] Im operativen Netz werden unter anderem die Radar-Daten zu den Arbeitsplätzen der Lotsen transportiert, was zur besseren Anschaulichkeit in Abbildung 6 dargestellt ist. Diese Darstellung ist stark vereinfacht. Der tatsächliche Weg der Radar-Daten ist weitaus komplizierter, was jedoch für diese Arbeit irrelevant ist.

[6] Dies könnte in Zukunft gelockert werden. Beispielweise gibt es Pläne, die Lotsen-Arbeitsplätze um Tablet-Computer zu ergänzen, und diese zunächst statisch, später vielleicht sogar per WLAN, mit aktuellen Informationen zu versorgen. Da solche Pläne aber mit den Ansprüchen an Sicherheit vereinbart werden müssen, vor allem der Ausfallsicherheit und dem Schutz vor Manipulation, ist dies alles noch Zukunftsmusik und spielt für das BNL-Modul noch keine Rolle.

[7] Genau genommen befinden sich diese Server nicht direkt im BK-Netz, sondern in einem von einer Firewall abgetrennten Bereich. Dies ist in Abbildung 6 bewusst vereinfacht dargestellt, da hier nicht die komplette Sicherheitsinfrastruktur der DFS abgebildet werden soll, sondern nur die Aspekte, die für das BNL-Modul und das allgemeine Verständnis wichtig sind.

Externe Stellen können ebenfalls unterschiedlichste Netzwerkstrukturen haben. Zudem muss davon ausgegangen werden, dass diese im Gegensatz zu den Antragstellern nicht bereit sein werden, spezielle Client-Software zu verwenden.

Für das BNL-Modul kann ein interner Server innerhalb des BK-Netzes (mit-)genutzt werden. Ebenfalls kann ein externer Server eingerichtet werden, der direkt im Internet erreichbar ist, sich dann jedoch außerhalb des BK-Netzes befinden muss. Ob dieser trotzdem physikalisch in den Gebäuden der DFS steht, oder von einem externen Dienstleister betrieben wird, ist derzeit nicht festgelegt.

Neben all diesen Möglichkeiten bestehen natürlich weiterhin die klassischen Kommunikationswege über Telefon und Fax.

Somit gibt es *nur* die folgenden Ansatzpunkte für die Software des BNL-Moduls:

- Interne Clients im BK-Netz auf den Arbeitsplatzrechnern der Supervisor und Sachbearbeiter
- Externe Clients auf den Arbeitsplatzrechnern der Antragsteller
- Interner Server im BK-Netz
- Externer Server im Internet, außerhalb des BK-Netzes

Diese Ansatzpunkte sind in Abbildung 6 hervorgehoben.

(basiert auf eigenen Recherchen)

Anforderung 42 (Ressourcenverbrauch – *nichtfunktional, notwendig*)
Der Ressourcenverbrauch des BNL-Moduls soll niedrig genug sein, dass serverseitig kein verteiltes System notwendig ist.

Das heißt, sowohl für den internen Server als auch für den externen Server (sofern vorhanden, wie in Anforderung 41 definiert) soll jeweils maximal eine virtuelle Maschine in den folgenden Dimensionen benötigt werden:

CPU-Kerne	CPU	RAM	Festplatte
1	1 GHz	1 GiB	5 GiB

Auch STANLY_ACOS läuft auf nur einem einzigen Server und lastet diesen bei weitem nicht aus,[8] obwohl an STANLY_ACOS bislang keine außergewöhnlichen Optimierungen vorgenommen wurden. Daher sollte sich der Ressourcenverbrauch des BNL-Moduls ebenfalls in Grenzen halten.

(basiert auf eigenen Recherchen)

Anforderung 43 (Browser – *nichtfunktional, notwendig*)
Wenn für die internen Clients (siehe Anforderung 41) ein Web-Client zum Einsatz kommt, soll dieser im Microsoft Internet Explorer ab Version 9.0.8112.16421 vollständig lauffähig sein.

(basiert auf eigenen Recherchen)

[8] Genau genommen sind es zwei Server, für die Hochverfügbarkeit. Jedoch ist stets nur einer von beiden aktiv.

Anforderung 44 (Sensible Daten – *nichtfunktional, notwendig*)
Sämtliche Daten der BNL-Vorhaben sollen stets im BK-Netz (definiert in Anforderung 41) verfügbar sein. Sie sollen nicht dauerhaft außerhalb des BK-Netzes gespeichert werden.

Zwar unterliegen diese Daten keiner Geheimhaltungspflicht, jedoch muss der Datenschutz gegenüber den Auftraggebern gewahrt bleiben. Zudem müssen sich Sachbearbeiter und Supervisor darauf verlassen können, dass die Informationen zu den BNL-Vorgaben innerhalb der DFS stets verfügbar sind.

(basiert auf eigenen Recherchen)

2.2. Wünschenswert

Anforderung 45 (Eingabe des Staffelungsbereiches – *funktional, wünschenswert*)
Zu jeder Geometrie soll ein Staffelungsbereich angegeben werden können. Dabei handelt es sich um einen Puffer um die eigentliche Geometrie herum. Folgende Maßeinheiten sollen unterstützt werden:

Kürzel	Einheit	Umrechnung
m	Meter	—
NM	Nautische Meilen	1NM = 1852m [Bur06, S. 127]

Die Voreinstellung soll NM sein.

Im Gegensatz zu der ursprünglichen Anforderung wurde die Einheit ft (Fuß) entfernt, da diese in der Flugsicherungs-Domäne nur für Höhenangaben benutzt wird.

(basiert auf eigenen Recherchen und BNL EIN 14 P: Eingabe des Staffelungsbereich [Har12, S. 27])

Anforderung 46 (Eingabe der Höhe – *funktional, wünschenswert*)
Diese Anforderung wurde aus [Har12] nicht übertragen, da sie für den Inhalt dieser Diplomarbeit keine Relevanz hat.

(basiert auf BNL EIN 15 P: Eingabe der Höhe [Har12, S. 27])

Anforderung 47 (Vorschau der Koordinaten – *funktional, wünschenswert*)
Beim Anlegen eines neuen Vorhabens soll es möglich sein, zunächst nur die Koordination einzutragen. Diese sollen in einer Vorschau vor einer Hintergrundkarte angezeigt werden.

Werden darüber hinaus noch die Zeitbereiche (Datum/Uhrzeit) eingetragen, sollen alle bestehenden Vorhaben in der Karte angezeigt werden, die das neue Vorhaben räumlich und zeitlich überlappen.

So kann der Nutzer im Voraus entscheiden, ob es sich lohnt, weitere Daten einzugeben.

(basiert auf BNL EIN 16 P: Vorschau der Koordinaten [Har12, S. 28])

Anforderung 48 (Supervisorreminder – *funktional, wünschenswert*)
Wird ein Vorhaben vom Typ *Gliding* (Segelflug) von einem Supervisor aktiviert, muss er dies an verschiedene Stellen melden:

Kürzel	Bedeutung
FIS	Flight Information Service
STCA	Short Time Conflict Alert
Assistant	Datenassistenten
TWR	Tower
Sectors	Nachbarsektoren / Adjacent Sectors

Als Erinnerung, welche Meldungen schon erfolgt sind, soll er in einem Segelflug-Vorhaben für jede dieser Stellen einen entsprechenden Vermerk setzen können, am besten über einfache Checkboxen direkt in der Vorhabensliste.

(basiert auf BNL TAB 10 P: Supervisorreminder für Segelflugsektoren [Har12, S. 36])

Anforderung 49 (Offene Schnittstelle – *nichtfunktional, wünschenswert*)
Die in Anforderung 14 beschriebene Direkteingabe soll nicht auf das BNL-Modul beschränkt sein, sondern auch andere Client-Software erlauben.

Einige Unternehmen erstellen schon jetzt ihre BNL-Anträge halbautomatisch durch ihre eigene Software. Diesen würde die Direkteingabe nicht entgegen kommen, da sie nun jeden BNL-Antrag von Hand in ein anderes Programm bzw. Webinterface übertragen müssten.

Eine stabile, offene Schnittstelle würde es diesen Unternehmen erlauben, ihre eigene Software so anzupassen, dass sie die BNL-Anträge direkt an das BNL-Modul sendet.

Die weitere Bearbeitung dieser Anträge erfolgt dann genau so, als wären sie von Hand über die Direkteingabe eingereicht worden.

(basiert auf eigenen Recherchen)

2.3. Optional

Anforderung 50 (Integration in STANLY_ACOS – *funktional, optional*)
Das BNL-Modul soll über einen entsprechenden Menüpunkt in STANLY_ACOS erreichbar sein.

Im Gegensatz zu der ursprünglichen Anforderung erhält diese Anforderung eine deutlich niedrigere Priorität (*optional* statt *notwendig*), um keine technologischen Erwägungen vorweg zu nehmen: Sollte eine Architektur gewählt werden, in der das BNL-Modul unabhängig von STANLY_ACOS existiert, so wäre dieser Menüpunkt vollkommen überflüssig.

(basiert auf eigenen Recherchen und BNL EIN 1 M: Menüführung in STANLY_ACOS 2.0 [Har12, S. 14])

Anforderung 51 (Weitere Import-Formate – *funktional, optional*)
Der in Anforderung 8 geforderte Import soll auch für weitere Formate möglich sein, etwa:

1. XLS-Format (Microsoft Excel)
2. Einfaches XML-Format
3. AIXM-Format

Für das XLS-Format wäre die erwartete Tabellenstruktur noch zu definieren. Für das einfache XML-Format müsste die Struktur ebenfalls noch definiert werden. Das AIXM-Format ist ein definierter, offener XML-Standard, allerdings ist dieser sehr komplex.
(basiert auf eigenen Recherchen)

Anforderung 52 (Messen in der Karte – *funktional, optional*)
Diese Anforderung wurde aus [Har12] nicht übertragen, da sie für den Inhalt dieser Diplomarbeit keine Relevanz hat.
(basiert auf BNL MAP 5 O: Messen [Har12, S. 39])

Anforderung 53 (Drucken der BNL-Tabelle – *funktional, optional*)
Diese Anforderung wurde aus [Har12] nicht übertragen, da sie für den Inhalt dieser Diplomarbeit keine Relevanz hat.
(basiert auf BNL PRI 2 O: Drucken des Bookingtable [Har12, S. 47])

Anforderung 54 (NOTAM-Erstellung – *funktional, optional*)
Diese Anforderung wurde aus [Har12] nicht übertragen, da sie für den Inhalt dieser Diplomarbeit keine Relevanz hat.
(basiert auf BNL NOT 1 O: Erstellen eines NOTAMs [Har12, S. 47])

Anforderung 55 (Anzeige militärischer Luftraum-Buchungen – *funktional, optional*)
In der Karte soll es einen weiteren Layer für die militärischen Luftraum-Buchungen geben.
Diese werden bereits in STANLY_ACOS verwaltet und stehen daher mitsamt ihrer geometrischen Grunddaten im Prinzip zur Verfügung.
Dieser Karten-Layer soll per Vorstellung ausgeblendet sein.
(basiert auf eigenen Recherchen)

3. Lösungs-Architekturen

Auf Basis der Anforderungen an das BNL-Modul in Kapitel 2 werden nun mögliche Software-Architekturen entworfen.

Doch was ist das eigentlich, die *Architektur* einer Software? Diese Frage muss zuallererst geklärt werden, im Abschnitt 3.1.

Im Abschnitt 3.2 wird dann die Architektur von STANLY_ACOS näher beleuchtet, denn dies ist eine vorhandene Applikation, die bereits viele Funktionalitäten umsetzt, die auch das BNL-Modul haben soll.

Nach diesen Vorbereitungen beginnt in Abschnitt 3.3 die Suche nach sinnvollen Architekturen. Dabei grenzen die *notwendigen* Anforderungen den Lösungsraum ein, in dem überhaupt nach sinnvollen Architekturen gesucht werden kann. Wie es bei Architekturen typisch ist, kommen hier vor allem die nichtfunktionalen Anforderungen zum Tragen, genauer gesagt:

- Anforderung 4 (Zeitbereichseingabe)
- Anforderung 8 (Koordinaten-Eingabe)
- Anforderung 14 (Direkteingabe durch Antragsteller)
- Anforderung 39 (Nutzertypen)
- Anforderung 41 (Netzwerkstruktur)
- Anforderung 42 (Ressourcenverbrauch)
- Anforderung 44 (Sensible Daten)

Innerhalb dieses Lösungsraums bieten die übrigen Anforderungen, d.h. die *wünschenswerten* und die *optionalen*, einen Maßstab zum Vergleich dieser Architekturen, um am Ende eine möglichst gute auszuwählen und umzusetzen.

Dies klingt zunächst nach einem mathematischen Optimierungsproblem, hat aber einen entscheidenen Unterschied: Der Wert einer Architektur lässt sich nicht ohne Weiteres auf eine reelle Zahl abbilden, und selbst für zwei Architekturen im direkten Vergleich lässt sich aufgrund unterschiedlich gelagerter Vor- und Nachteile nicht immer auf Anhieb sagen, welche die bessere Architektur ist.

Daher wird in dieser Arbeit, wie es generell beim Entwurf von Software-Architekturen üblich ist, zunächst ein möglichst großes Spektrum an Architekturen betrachtet, das dann auf einzelne, besonders aussichtsreiche Kandidaten einschränkt wird. Nur diese werden weiter ausgearbeitet, um sie später in Kapitel 4 zu evaluieren.

3.1. Definition

Eine exakte Definition von *Architektur* bei Software ist in der Fachliteratur nur schwer zu finden, da es unterschiedliche Meinungen darüber gibt, was genau dazu gehört und

was nicht. Zudem hat sich die Bedeutung des Begriffes mit der Zeit gewandelt.

In den 1970er Jahren definierte Brooks [Bro75, S. 45] die Architektur als detaillierte Beschreibung der Schnittstelle zum Benutzer. Für ihn war Architektur einfach das Gegenstück zur Implementierung. Die Architektur solle das *Was* klären und die Implementierung das *Wie*. Der Begriff war so weit gefasst, dass bereits ein ausführliches Benutzerhandbuch als Architektur-Beschreibung galt, oder die Spezifikation einer Programmiersprache als Architektur des zugehörigen Compilers.

In der heutigen Zeit hat sich der Begriff weiter ausdifferenziert: Architektur ist, grob gesagt, die Beschreibung eines Software-Systems aus der Vogelperspektive. Sie ist die erste Phase eines jeden Entwurfsprozesses und soll klären, welches die wichtigstens Komponenten des Systems sind, welche davon selbst entwickelt werden und welche durch fertige Standardkomponenten realisiert werden. Weiterhin legt sie fest, welche Komponenten miteinander kommunizieren, und noch wichtiger, welche nicht.[1] Unter den vorhandenen Kommunikationswegen soll sie die Schnittstellen grob beschreiben, zwischen den Komponenten wie auch nach außen hin. Zudem wird die Architektur mehr und mehr als soziales Konstrukt und weniger als rein technische Beschreibung verstanden. Dabei soll die Architektur nicht zu sehr in die Details des Designs gehen, und (wie auch schon in der ursprünglichen Bedeutung) erst recht keine Implementierungsdetails behandeln. Bis zu welchen Detailgrad jedoch eine Architektur vordringen darf, und an welchen Stellen sie wie weit gehen darf, darüber gibt es abweichende Meinungen und dementsprechend viele Versuche einer Definition.

Im Rahmen dieser Arbeit soll der Definitions-Vorschlag von Johnson gelten, den Fowler in [Fow03] zitierte und dadurch bekannt machte:[2]

> „In den meisten erfolgreichen Softwareprojekten haben die Chefentwickler ein gemeinsames Bild von dem Systemdesign. Dieses gemeinsame Bild wird *Architektur* genannt. Es beinhaltet, wie das System in Komponenten unterteilt ist und wie diese Komponenten durch ihre Schnittstellen hindurch aufeinander einwirken. Üblicherweise setzen sich diese Komponenten wiederum aus kleineren Komponenten zusammen, aber die Architektur beinhaltet nur die Komponenten und Schnittstellen, über die *alle* Entwickler im Bilde sind."

3.2. STANLY_ACOS

Da es sich beim BNL-Modul um eine – wie auch immer geartete – Ergänzung zu STANLY_ACOS handelt, soll an dieser Stelle zunächst einmal die Architektur von STANLY_ACOS näher beleuchtet werden. STANLY_ACOS ist eine Webapplikation mit einer 3-Schichten-Architektur bestehend aus Persistenzschicht, Mittelschicht und Client-

[1] Dies zu erzwingen ist ein äußerst wichtiges Sicherheitsprinzip: *Enforcing explicit data flow* [Ber07, S. 5]

[2] Original: „In most successful software projects, the expert developers working on that project have a shared understanding of the system design. This shared understanding is called 'architecture.' This understanding includes how the system is divided into components and how the components interact through interfaces. These components are usually composed of smaller components, but the architecture only includes the components and interfaces that are understood by all the developers."

Schicht, wie in Abbildung 7 dargestellt.

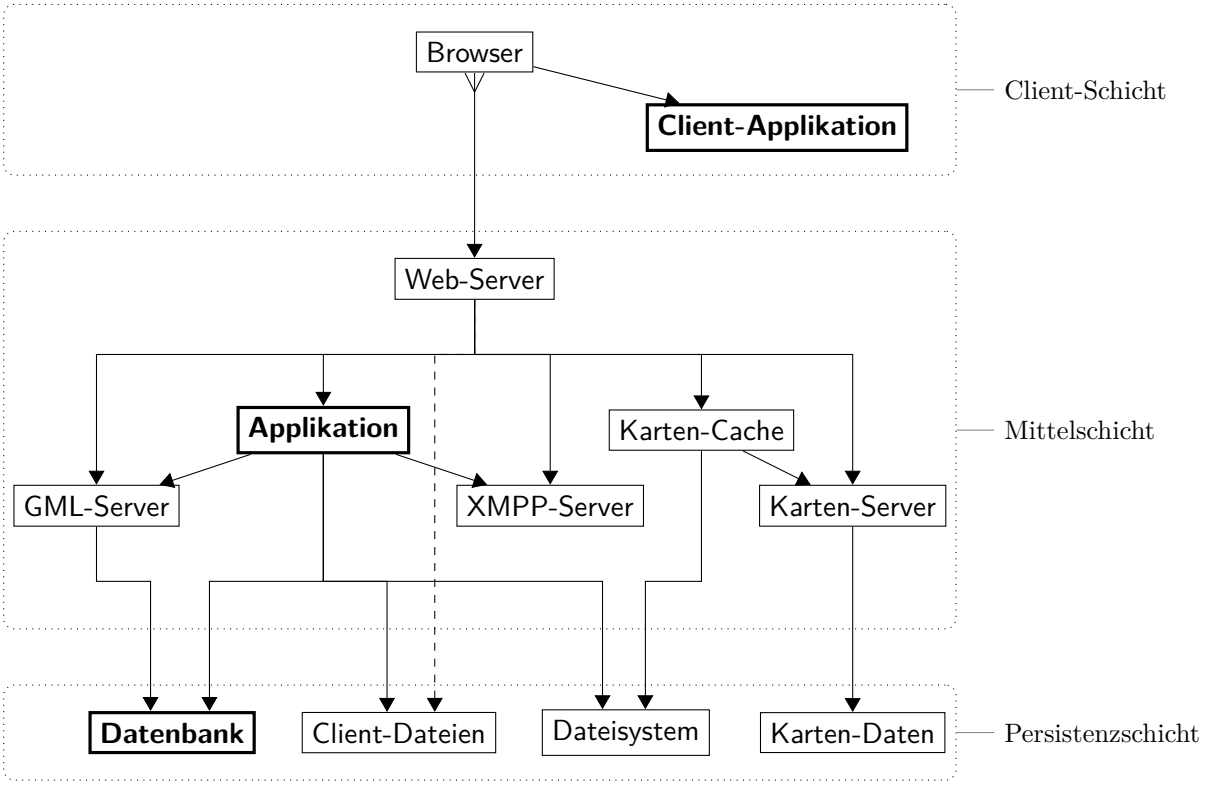


Abbildung 7.: Architektur von STANLY_ACOS. Hervorgehoben sind die Komponenten, die keine reinen Standardkomponenten sind. Ihr innerer Aufbau wird in den folgenden Abbildungen gezeigt. Der gestrichelte Kommunikationsweg steht aus Performance-Gründen nur im Debug-Modus zur Verfügung. Die Notation ist in Abschnitt 1.8 erklärt.

Diese Architektur unterscheidet sich jedoch von der klassischen 3-Schichten-Architektur von Webapplikationen in einigen wesentlichen Punkten:

1. Die Mittelschicht besteht nicht nur aus der eigentlichen Applikation, sondern enthält weitere Standardkomponenten wie zum Beispiel einen Karten-Server für die Hintergrundkarte und einen XMPP-Server[3] für serverseitig initiierte Benachrichtigungen an alle Clients. All diese Komponenten werden über den Web-Server, der zugleich als Reverse Proxy fungiert, in einem gemeinsamen URL-Schema zur

[3]XMPP steht für *Extensible Messaging and Presence Protocol* und wurde früher *Jabber* genannt. Es kann auf effiziente Art und Weise über HTTP transportiert werden, sodass es für im Browser laufende JavaScript-Applikationen erreichbar ist, ohne dass diese ständig ein Polling durchführen müssen. Diese Technik wird BOSH (Bidirectional-streams Over Synchronous HTTP) genannt und ist in XEP-0124 [PSSAM10] in Kombination mit XEP-0206 [PSA10] spezifiziert.

Verfügung gestellt (siehe Abbildung 7).

2. Der Client ist eine eigenständig im Browser laufende JavaScript-Anwendung und damit eine Rich Internet Application (RIA). Die Kommunikation mit der Serverseite erfolgt über asynchron nachgeladene strukturierte JSON-Daten.
3. Die Geschäftsregeln (Business Rules) sind datenbankseitig implementiert, durch entsprechende Constraints und Checkfunktionen, die gegebenenfalls von benutzerdefinierten Datenbankfunktionen Gebrauch machen. So wird die vollständige Integrität bereits datenbankseitig sichergestellt. Auswerteroutinen und Aggregationen (auch hierarchische) liegen ebenfalls überwiegend in der Datenbank, in Form von Views oder bei Bedarf auch benutzerdefinierten Datenbankfunktionen (siehe Abbildung 8).

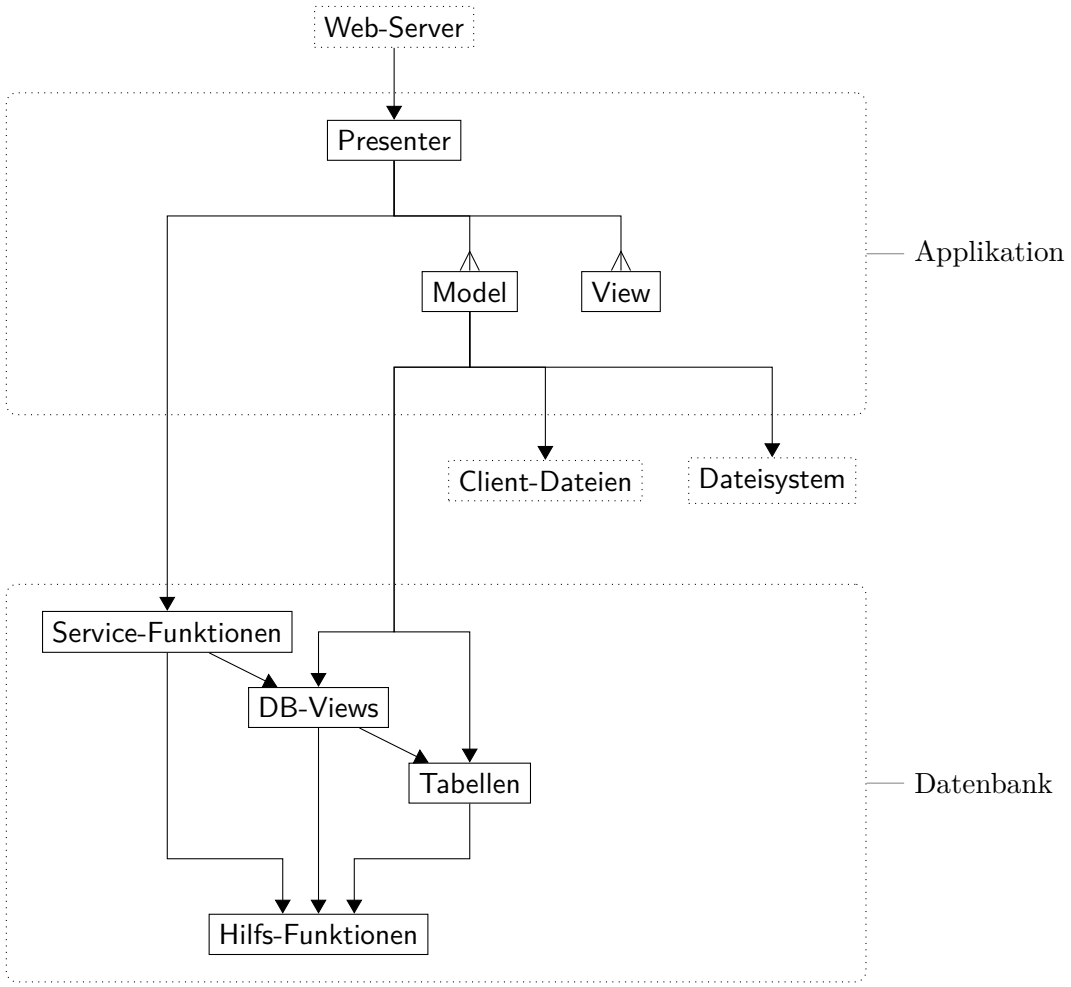


Abbildung 8.: Komponenten *Applikation* und *Datenbank* von STANLY_ACOS im Zusammenhang. Die Notation ist in Abschnitt 1.8 erklärt.

Durch die erste Architekturentscheidung erscheint die Mittelschicht relativ groß. Doch durch die letzten beiden Architekturentscheidungen ist die in der Mittelschicht enthaltene Applikation relativ klein und kümmert sich im Wesentlichen nur noch um die Aufgaben, die von der Datenbank nicht ohne Weiteres übernommen werden können.[4] Dennoch ist die Applikation intern via Model-View-Presenter (MVP)[5] organisiert, auch wenn der *Model*-Bereich sehr viel Arbeit an die Datenbank delegiert und der *View*-Bereich fast alles an den Client. Eine objektrelationale Abbildung (ORM) wird nur an sehr wenigen Stellen benötigt.

Oft ist die Applikation sogar lediglich damit beauftragt, Anfragen des Clients zu dekodieren und direkt an die Datenbank weiterzureichen, um dann die Antwort der Datenbank zurück an den Client durchzuleiten. In diesen Fällen erfolgt dies direkt im *Presenter*-Bereich und es kommen weder *Models* noch *Views* zum Einsatz (siehe Abbildung 8). Die Antwort-Daten werden dabei bereits datenbankseitig strukturiert, aggregiert und serialisiert.

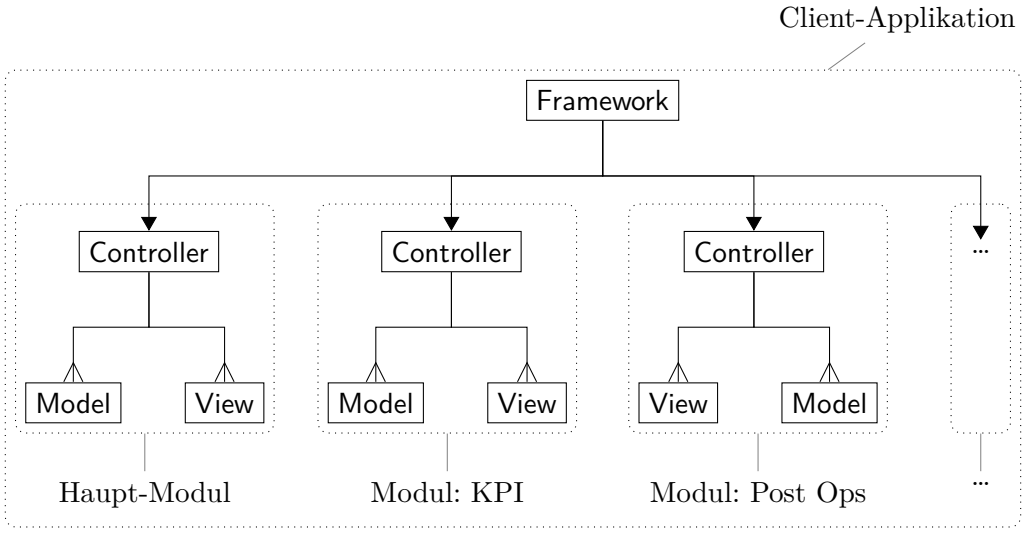


Abbildung 9.: Komponente *Client-Applikation* von STANLY_ACOS. Die Notation ist in Abschnitt 1.8 erklärt.

[4] Dies sind zum Beispiel das Bedienen des HTTP-Protokolls, die Session-Verwaltung, das Durchsetzen von Authentifikation/Autorisation, das Parsen von Daten aus externen Quellen und das Generieren von Excel-Tabellen.

[5] MVP ist eine gängige Abwandlung von MVC (Model-View-Controller) bei Webanwendungen. Das MVC-Konzept stammt ursprünglich aus der Smalltalk-Welt und kommt bis heute in den großen GUI-Frameworks wie GTK und Qt zum Einsatz. In klassischen Webanwendungen ist eine direkte Umsetzung des MVC-Konzepts kaum sinnvoll, da jedes kleine GUI-Ereignis im Browser einen Round Trip zum Server benötigen würde. So hat sich MVP als Abwandlung von MVC etabliert, auch wenn zahlreiche MVP-Frameworks sich als *MVC* bezeichneten und damit eine Verwässerung des Begriffes einleiteten. Erst mit dem Aufkommen größerer JavaScript-Anwendungen im Browser etablierte sich wieder das klassische MVC in Webanwendungen, allerdings in der obersten Schicht (Client) statt in der Mittelschicht.

Die Client-Applikation ist intern als klassische GUI-Anwendung nach dem Muster Model-View-Controller (MVC) aufgebaut. Genauer gesagt besteht sie aus mehreren Modulen, die jeweils per MVC strukturiert sind (siehe Abbildung 9).

Die Client-Applikation wird nicht direkt durch den Web-Server, sondern durch die serverseitige Applikation ausgeliefert, die alle Dateien des Clients zusammenfasst, minimiert und komprimiert. Das Ergebnis ist eine einzelne gezippte HTML-Datei, die für weitere Zugriffe gecached wird, bis sich etwas an den Dateien der Client-Applikation ändert. Dennoch gibt es für Debugging-Zwecke auch die Möglichkeit, den Client in seiner originalen Form direkt vom Web-Server ausliefern zu lassen (siehe Abbildung 7).

3.3. Hauptkomponenten

Nun beginnt die Suche nach geeigneten Architekturen.

Die Architekturen richten sich zunächst einmal an der Hardware aus, denn die Teile der Software, die auf unterschiedlichen Rechnern laufen, sind zwangsläufig separate Komponenten, die über ein geeignetes Protokoll miteinander kommunizieren müssen. Nach Anforderung 41 (Netzwerkstruktur) gibt es nur vier verschiedene Orte, an denen Software des BNL-Moduls laufen kann. Die entsprechenden Software-Komponenten, die gewissermaßen die Hauptkomponenten darstellen, sollen die folgenden Namen tragen:

Interner Client – der Teil der Software, der auf den Arbeitsplatz-Rechnern im BK-Netz läuft, egal ob im Browser oder als separate Software. Dies ist der Teil, über den die Supervisor und die Sachbearbeiter mit dem BNL-Modul in Verbindung treten.

Interner Server – der Teil der Software, der auf einem[6] Server im BK-Netz der DFS läuft.

Externer Client – der Teil der Software, der auf den Arbeitsplatz-Rechnern der Antragsteller läuft, egal ob im Browser oder als separate Software.

Externer Server – der Teil der Software, der auf einem[7] Server außerhalb der DFS-internen Netze läuft, direkt zugänglich via Internet.

Das sind die Möglichkeiten. Doch *muss* an all diesen Stellen auch Software laufen? Zunächst einmal *muss* es wegen Anforderung 39 (Nutzertypen) einen internen Client geben, da sonst weder die Supervisor noch die Sachbearbeiter Zugriff auf das BNL-Modul hätten. Auch *muss* es laut Anforderung 14 (Direkteingabe durch Antragsteller) einen externen Client geben.[8]

[6] Laut Anforderung 42 (Ressourcenverbrauch) soll es maximal einen internen Server geben.

[7] Laut Anforderung 42 (Ressourcenverbrauch) soll es maximal einen externen Server geben.

[8] Dies bedeutet natürlich nicht, dass jeder Antragsteller den externen Client tatsächlich auch nutzen muss. Dieser richtet sich vor allem an die Vielnutzer, während zusätzlich nach wie vor die bisherigen Wege via (unstrukturierter) E-Mail, Fax und Telefonanruf zur Verfügung stehen.

Jedoch gibt es keine Anforderung, die einen internen oder externen Server erzwingt. Dies liefert bereits 4 Möglichkeiten:[9]

BS-* – beide Server
ES/* – nur externer Server
IS/* – nur interner Server
KS/* – kein Server

Schaut man sich zusätzlich die Schnittstellen zwischen den Komponenten an, so sind diese recht eindeutig: Jeder Client kommuniziert mit dem entsprechenden Server. Nur in dem Fall, dass es zwei Server gibt, einen internen wie auch einen externen, stellt sich zusätzlich die Frage, welche der internen Komponenten mit dem externen Server kommuniziert. Somit gibt es insgesamt die folgenden 6 Möglichkeiten:

BS-* – Es gibt beide Server, einen internen wie auch einen externen.

- **BS-IS/*** – nur der interne Server kommuniziert mit dem externen Server
- **BS-IC/*** – nur der interne Client kommuniziert mit dem externen Server
- **BS-B/*** – beide kommunizieren mit dem externen Server

ES/* – Es gibt nur einen extern Server.
IS/* – Es gibt nur einen intern Server.
KS/* – Es gibt keinen Server, weder intern noch extern.

Diese Architekturen werden in den folgenden Unterabschnitten weiter ausgearbeitet, wobei sich BS-B/* und KS/* als fruchtlos erweisen, während die übrigen Architekturen ein reichhaltiges Angebot liefern. Bei letzteren Architekturen stellt sich jeweils die Frage, welche der Hauptkomponenten durch eine Standardkomponententente und welche durch Eigenentwicklung realisiert werden sollen.

Mit *Standardkomponente* ist dabei eine fertige Softwarekomponente gemeint, die lediglich konfiguriert wird. Muss diese hingegen mit zusätzlichem Programmcode ausgestattet werden (z.B. benutzerdefinierte Datenbankfunktionen, oder ein Wrapper um die Komponente herum), so zählt das im Rahmen dieser Arbeit nicht als Standardkomponente, sondern als Implementierungsdetail einer selbstentwickelten Komponente. Auch wenn mehrere Standardkomponenten kombiniert werden, zählt dieses komplexe Zusammenspiel als Implementierungsdetail einer selbstentwickelten Komponente.[10] Es geht also bei einer Standardkomponente um die direkte Anwendbarkeit einer fertigen, leicht integrierbaren, gut getesteten Software, denn dies stellt nach [Bro95, S. 197–199] nicht nur einen quantitativen Vorteil dar (man spart sich Entwicklungsarbeit), sondern einen qualitativen Vorteil! Zudem unterscheidet sich laut [Bro95, S. 4–6] ein frisch entwickeltes, korrekt

[9] Die Endungen -* und /* bedeuten, dass diese Architektur-Möglichkeiten nachfolgend weiter unterteilt werden. Zum Beispiel wird Architektur IS/* später in IS/E-∅-EN-ER, IS/E-∅-EN-EK, … unterteilt. Architektur BS-* wird zunächst in die Architekturen BS-IS/*, BS-IC/*, … unterteilt, die dann nochmals in BS-IS/E-E-EN-EN, BS-IS/E-E-EN-ER, … unterteilt werden.

[10] Einzige Ausnahme sind die Komponenten eines Mailservers, d.h. die Kombination aus SMTP- und POP/IMAP-Server. Deren nahtloses Zusammenspiel ist so allgegenwärtig, dass ein Mailserver in diesem Zusammenhang als Ganzes gesehen werden soll, d.h. als eine einzelne Standardkomponente.

funktionierendes Programm von einem marktfähigen, integrierbaren Produkt etwa um den Faktor 9 im Entwicklungsaufwand.

In dieser Arbeit werden folgende Server-Komponenten untersucht:

S* – Standardkomponente

- **SD** – Datenbank (PostgreSQL, MariaDB, …)
- **SH** – via HTTP erreichbare dokumentenorientierte Datenbank bzw. transaktionsfähiger WFS-Server[11] (CouchDB, Deegree, …)
- **SM** – Mailserver (Postfix+Dovecot, Courier, …)
- **SX** – XMPP-Server (ejabberd, Prosody, …)

E – Eigenentwicklung

Für die Clients gibt es keine Standardkomponenten, da die Anforderungen an die GUI sehr speziell sind.[12] Stattdessen soll für die Client-Komponenten der Applikations-Typ der Eigenentwicklung unterschieden werden:

S* – Standardkomponente

- **(keine)**

E* – Eigenentwicklung

- **EN** – Nativer Client (C++/Qt, Java/SWT, …)
- **ER** – Rich Internet Application (JavaScript/ExtJS, CoffeeScript/jQuery, …)
- **EK** – Klassische Webapplikation (Client-Logik auf dem Server, Auslieferung von HTML-Seiten, JavaScript nur unterstützend)

Aus den folgenden Unterabschnitten wird hervor gehen, dass diese Unterscheidung tatsächlich sinnvoll ist, da diese Applikations-Typen unterschiedliche Einbettungs- und Kommunikationsmöglichkeiten bieten.

Nach der Ausarbeitung der Architekturen folgt in Abschnitt 3.4 eine Zusammenfassung, welche Kombinationen von Hauptkomponenten in welchen Architekturen möglich sind.

[11] WFS (Web Feature Service) ist ein auf HTTP basiertes Protokoll zum Übertragen von geographischen Vektordaten, die mit (ggf. hierarchisch strukturierten) Metadaten angereichert sind. Auch reine Metadaten ohne Vektordaten sind möglich. Daher ist ein WFS-Server vom Konzept her eine dokumentenorientierte Datenbank, wenn auch eine recht spezielle. Ein *transaktionsfähiger WFS-Server* beherrscht neben den Abfrage-Operationen auch Operationen zum Einfügen, Aktualisieren und Löschen von Kartenobjekten. [Ope10b]

[12] Zum Beispiel deutet Anforderung 4 (Zeitbereichseingabe) auf eine Kalender- bzw. Zeitplanungs-Applikation hin, während in Anforderung 8 (Koordinaten-Eingabe) eher ein Geoinformationssystem (GIS) verlangt wird.

3.3.1. Architektur BS-IS/*

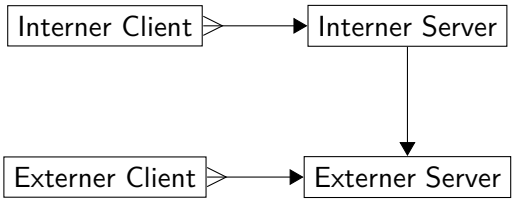


Abbildung 10.: Architektur BS-IS/*. Die Notation ist in Abschnitt 1.8 erklärt.

Wie in Abbildung 10 dargestellt gibt es einen internen und einen externen Server, und nur der interne Server, nicht der interne Client, kommuniziert mit dem externen Server.

Der interne Client muss effizient auf die Daten des internen Servers zugreifen können (z.B. durch Datenbank-Indizes), daher kann der interne Server kein Mailserver oder XMPP-Server (SM, SX) sein. Andererseits muss der interne Server aber die angefallenen Daten automatisch vom externen Server holen, wodurch eigentlich nur ein Mailserver oder XMPP-Server (SM, SX) in Frage kommen. Somit bleibt für den internen Server nur noch die Eigenentwicklung (E).

Für den internen Client gibt es entsprechend keine Einschränkungen, alle 3 Varianten (EN, ER, EK) kommen in Frage.

Für den externen Server kommen alle Varianten bis auf die Datenbank (SD) in Frage, da diese vom internen Server weder per SMTP noch über den HTTP-Proxy erreichbar wäre, und diese Kommunikationswege laut Anforderung 41 (Netzwerkstruktur) respektiert werden sollen.

Ist der externe Server eine Eigenentwicklung, so gibt es keine Einschränkung an den externen Client (EN, ER, EK).

Ist er hingegen eine Standardkomponente, so kann der externe Client kein klassischer Webclient (EK) mehr sein, da sich ein klassischer Webclient dadurch auszeichnet, dass die gesamte Logik auf Serverseite implementiert ist.

Ist der externe Server zudem ein Mailserver, so kann der externe Client auch keine Rich Internet Application (ER) sein, da die Möglichkeiten zum Versand von E-Mails aus dem Browser heraus sehr beschränkt sind, und der direkte Empfang sogar unmöglich ist. Somit kommt für den externen Client in diesem Fall nur ein nativer Client (EN) in Frage.

Zusammengefasst gibt es damit folgende Möglichkeiten für die Auswahl der Komponenten:[13]

Arch.	Int. Server	Ext. Server	Int. Client	Ext. Client	#	Σ
BS-IS/*	E	E	EN,ER,EK	EN,ER,EK	9	
BS-IS/*	E	SH,SX	EN,ER,EK	EN,ER	12	
BS-IS/*	E	SM	EN,ER,EK	EN	3	24

[13]Die Spalte # enthält die Anzahl der Möglichkeiten pro Zeile. So zeigt die erste Zeile 3 Möglichkeiten des internen Clients, die frei mit den 3 Möglichkeiten des externen Clients kombiniert werden können, also insgesamt $3 \cdot 3 = 9$ Kombinationen. Die Spalte Σ enthält die Gesamt-Anzahl für die Architektur.

3.3.2. Architektur BS-IC/*

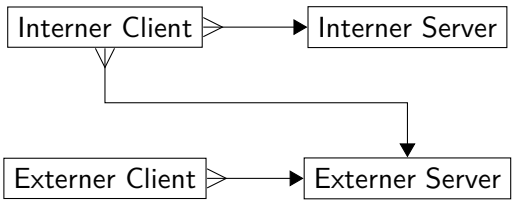


Abbildung 11.: Architektur BS-IC/*. Die Notation ist in Abschnitt 1.8 erklärt.

Wie in Abbildung 11 dargestellt gibt es einen internen und einen externen Server, und nur der interne Client, nicht der interne Server, kommuniziert mit dem externen Server.

Wie in BS-IS/* (Unterabschnitt 3.3.1) muss der interne Client effizient auf die Daten des internen Servers zugreifen können, daher kann der interne Server kein Mailserver oder XMPP-Server (SM, SX) sein. Im Gegensatz zu BS-IS/* kommen in dieser Architektur jedoch auch eine bloße Datenbank bzw. Dokumentendatenbank (SD, SH) für den internen Server in Frage.

Ebenfalls wie in BS-IS/* kommen für den externen Server alle Varianten bis auf die Datenbank (SD) in Frage, da diese vom internen Client nicht erreicht werden kann.

Der interne Client kann keine klassische Webapplikation (EK) sein, denn in diesem Fall wäre es letztendlich doch der interne Server, der die Kommunikation mit dem externen Server durchführt, was bereits in BS-IS/* behandelt wurde. Der interne Client kann also nur ein nativer Client (EN) oder eine Rich Internet Application (ER) sein.

Ist der externe Server eine Eigenentwicklung, so gibt es keine Einschränkung an den externen Client (EN, ER, EK). Auch an den internen Client stellt dieser keine Einschränkung, sodass der interne Client nur noch durch die Wahl des internen Servers eingeschränkt wird: Ist der interne Server eine Datenbank (SD) und keine Dokumentendatenbank, so ist dieser vom Browser via HTTP nicht erreichbar und der interne Client kann entsprechend keine Rich Internet Application (ER) sein.

Ist der externe Server eine Dokumentendatenbank oder ein XMPP-Server (SH, SX), so gilt für den internen Server und Client genau das Gleiche. Der einzige Unterschied ist, dass der externe Client kein klassischer Webclient (EK) mehr sein kann, da dies nur bei einer Eigenentwicklung des externen Server möglich ist.

Ist der externe Server jedoch ein Mailserver (SM), so können weder der interne noch der externe Client im Browser laufen, da dort der Versand von E-Mails nur sehr beschränkt möglich ist, und der direkte Empfang sogar unmöglich ist. Somit müssen in diesem Fall beide Clients nativ (EN) sein. Dies bedeutet aber interessanterweise, dass es für den internen Server hier keine zusätzlichen Einschränkungen gibt (E, SD, SH).

Zusammengefasst gibt es damit folgende Möglichkeiten für die Auswahl der Komponenten:

Arch.	Int. Server	Ext. Server	Int. Client	Ext. Client	#	Σ
BS-IC/*	E,SH	E	EN,ER	EN,ER,EK	12	
BS-IC/*	SD	E	EN	EN,ER,EK	3	
BS-IC/*	E,SH	SH,SX	EN,ER	EN,ER	16	
BS-IC/*	SD	SH,SX	EN	EN,ER	4	
BS-IC/*	E,SD,SH	SM	EN	EN	3	38

3.3.3. Architektur BS-B/* (verworfen)

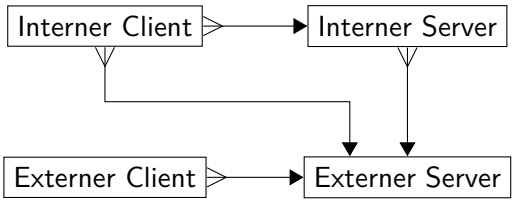


Abbildung 12.: Architektur BS-B/* (verworfen). Die Notation ist in Abschnitt 1.8 erklärt.

Wie in Abbildung 12 dargestellt gibt es einen internen und einen externen Server, und sowohl der interne Client als auch der interne Server kommunizieren mit dem externen Server.

Diese Architektur hat keinen einzigen Vorteil gegenüber der Architektur BS-IS/* (Unterabschnitt 3.3.1), führt aber ein unnötiges zusätzliches Risiko durch die geteilte Verantwortlichkeit zum Abholen der Daten vom externen Server ein.

Daher wird diese Architektur verworfen.

3.3.4. Architektur ES/*

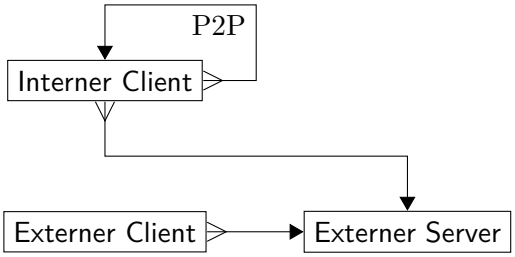


Abbildung 13.: Architektur ES/*. Die Notation ist in Abschnitt 1.8 erklärt.

Wie in Abbildung 13 dargestellt gibt es nur einen externen Server. Diesen dürfen die internen Clients wegen Anforderung 44 (Sensible Daten) allerdings nicht zur Speicherung ihrer Daten verwenden. Sie müssen sich daher selbst organisieren und ein Peer-to-Peer-Netzwerk bilden. Da dies mit aktuellen Browsern jedoch nicht möglich ist, müssen die

internen Clients native Clients (EN) sein.

Wie in BS-IS/* (Unterabschnitt 3.3.1) und BS-IC/* (Unterabschnitt 3.3.2) kommen für den externen Server alle Varianten bis auf die Datenbank (SD) in Frage, da diese von den internen Clients nicht erreicht werden kann.

Ebenfalls wie in BS-IC/* wird der externe Client nur noch durch die Wahl des externen Servers eingeschränkt: Eine Eigenentwicklung ermöglicht alle externen Clients (EN, ER, EK), eine Dokumentendatenbank oder ein XMPP-Server ermöglicht alle externen Clients bis auf die die klassische Webapplikation (EK), und ein Mailserver funktioniert nur in Kombination mit einem nativen Client (EN).

Zusammengefasst gibt es damit folgende Möglichkeiten für die Auswahl der Komponenten:[14]

Arch.	Int. Server	Ext. Server	Int. Client	Ext. Client	#	Σ
ES/*	∅	E	EN	EN,ER,EK	3	
ES/*	∅	SH,SX	EN	EN,ER	4	
ES/*	∅	SM	EN	EN	1	8

3.3.5. Architektur IS/*

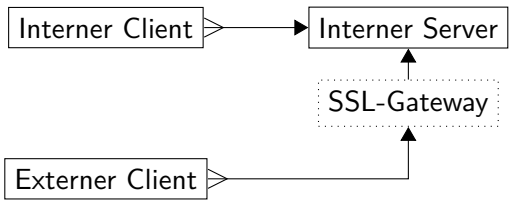


Abbildung 14.: Architektur IS/*. Die Notation ist in Abschnitt 1.8 erklärt.

Wie in Abbildung 14 dargestellt gibt es nur einen internen Server. Der externe Client muss sich mit dem internen Server verbinden, was nur über das SSL-Gateway möglich ist.

Dies ist die einzige Architektur, in der eine Rollenverwaltung nötig ist, um zwischen Supervisor/Sachbearbeiter und Auftraggeber, wie in Anforderung 39 (Nutzertypen) definiert, zu unterscheiden. In den übrigen Architekturen ist diese Trennung bereits dadurch gegeben, dass sich die Auftraggeber nur mit dem externen Server verbinden, den es hier in dieser Architektur nicht gibt.

Da das SSL-Gateway nur zur Nutzung über den Browser vorgesehen ist, und diese Kommunikationswege laut Anforderung 41 (Netzwerkstruktur) respektiert werden sollen, kann der externe Client kein nativer Client (EN) sein.

Zudem kann der interne Server keine Datenbank (SD) sein, da diese ebenfalls nicht über das SSL-Gateway erreichbar wäre. Wie in BS-IC/* (Unterabschnitt 3.3.2) muss der interne Client effizient auf die Daten des internen Servers zugreifen können, daher kann

[14]Eine nicht vorhandene Komponente wird durch die Leerstelle ∅ gekennzeichnet.

der interne Server auch kein Mailserver oder XMPP-Server (SM, SX) sein. Somit kann der interne Server nur eine Eigenentwicklung (E) oder eine Dokumentendatenbank (SH) sein.

Wie gehabt können der interne und externe Client nur dann klassische Webapplikationen (EK) sein, wenn der interne Server eine Eigenentwicklung (E) ist.

Zusammengefasst gibt es damit folgende Möglichkeiten für die Auswahl der Komponenten:

Arch.	Int. Server	Ext. Server	Int. Client	Ext. Client	#	Σ
IS/*	E	∅	EN,ER,EK	ER,EK	6	
IS/*	SH	∅	EN,ER	ER	2	8

3.3.6. Architektur KS/* (verworfen)

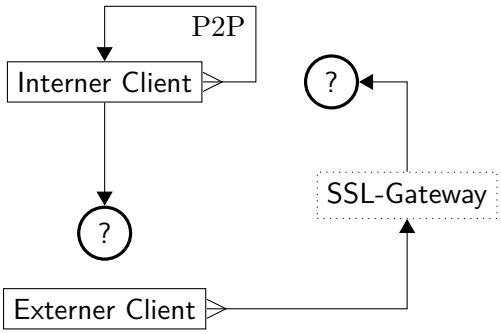


Abbildung 15.: Architektur KS/* (verworfen). Die Notation ist in Abschnitt 1.8 erklärt.

Wie in Abbildung 15 dargestellt gibt es weder einen internen noch einen externen Server. Das kann jedoch in der gegebenen Netzwerkstruktur (Anforderung 41) nicht funktionieren.

Die internen Clients könnten sich zwar wie in Architektur ES/* (Unterabschnitt 3.3.4) als Peer-to-Peer-Netzwerk organisieren. Doch sie haben keine Chance, die externen Clients irgendwo im Internet zu finden.

Also müssten die externen Clients eine Verbindung nach intern aufbauen, was nur über das SSL-Gateway möglich ist. Das SSL-Gateway hat jedoch unter den Clients im P2P-Netzwerk keinen festen Ansprechpartner und weiß daher nicht, wohin es Anfragen von externen Clients weiterleiten sollte.[15]

Somit wird diese Architektur verworfen.

[15] Anders gesagt: Wenn es einen solchen festen Ansprechpartner gäbe, hätte dieser die Rolle eines internen Servers. Somit wäre diese Architektur nicht KS/*, sondern IS/* (Unterabschnitt 3.3.5).

3.4. Zusammenfassung

Es ergeben sich 6 theoretisch denkbare Möglichkeiten für die Struktur der Hauptkomponenten, von denen jedoch 2 einer näheren Betrachtung nicht standhalten. Die verbliebenen 4 funktionierenden Strukturen sind in Abbildung 16 dargestellt.

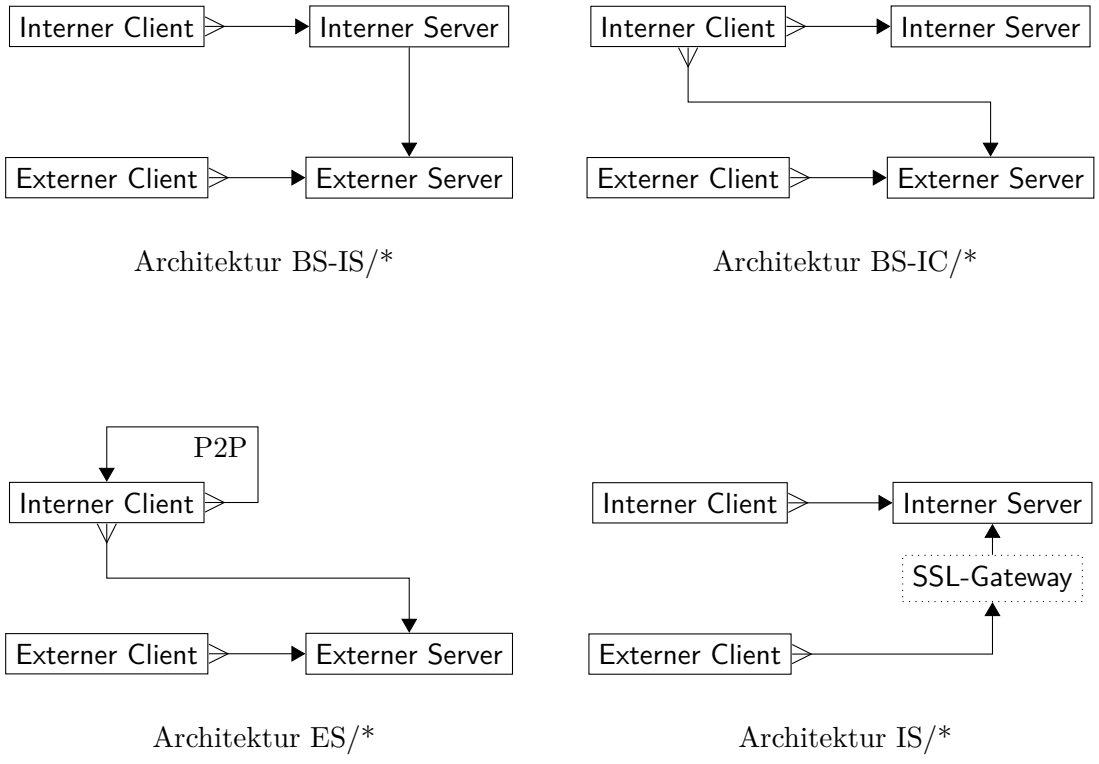


Abbildung 16.: Übersicht der funktionierenden Architekturen. Die Notation ist in Abschnitt 1.8 erklärt.

Für die jeweilige Bestückung dieser Strukturen mit verschiedenen Hauptkomponenten gibt es insgesamt 540 theoretisch[16] denkbare Möglichkeiten, von denen jedoch nur die folgenden 78 Kombinationen tatsächlich sinnvoll sind:[17]

Arch.	Int. Server	Ext. Server	Int. Client	Ext. Client	#	Σ
BS-IS/*	E	E	EN,ER,EK	EN,ER,EK	9	
BS-IS/*	E	SH,SX	EN,ER,EK	EN,ER	12	
BS-IS/*	E	SM	EN,ER,EK	EN	3	24
BS-IC/*	E,SH	E	EN,ER	EN,ER,EK	12	
BS-IC/*	SD	E	EN	EN,ER,EK	3	
BS-IC/*	E,SH	SH,SX	EN,ER	EN,ER	16	
BS-IC/*	SD	SH,SX	EN	EN,ER	4	
BS-IC/*	E,SD,SH	SM	EN	EN	3	38
ES/*	∅	E	EN	EN,ER,EK	3	
ES/*	∅	SH,SX	EN	EN,ER	4	
ES/*	∅	SM	EN	EN	1	8
IS/*	E	∅	EN,ER,EK	ER,EK	6	
IS/*	SH	∅	EN,ER	ER	2	8
						78

Die Abkürzungen bedeuten (genaue Erklärung in Abschnitt 3.3):

BS-IS/* – beide Server vorhanden, interner *Server* kommuniziert mit externem Server
BS-IC/* – beide Server vorhanden, interner *Client* kommuniziert mit externem Server
ES/* – nur externer Server vorhanden, intern Peer-to-Peer
IS/* – nur interner Server vorhanden, externer Zugriff via SSL-Gateway
SD – Standardkomponente: Datenbank
SH – Standardkomponente: Dokumenten-Datenbank oder trans. WFS-Server
SM – Standardkomponente: Mailserver
SX – Standardkomponente: XMPP-Server
E – Eigenentwicklung
EN – Eigenentwicklung: Nativer Client
ER – Eigenentwicklung: Rich Internet Application
EK – Eigenentwicklung: Klassische Webapplikation

All diese Architekturen erfüllen die Anforderungen der höchsten Priorität (*notwendig*).

[16] Es stehen jeweils 5 Serverkomponenten (E, SD, SH, SM, SX) und 3 Clientkomponenten (EN, ER, EK) zur Auswahl. Für BS-IS/* und BS-IC/* gibt es jeweils $5 \cdot 5 \cdot 3 \cdot 3 = 225$ mögliche Kombinationen für zwei Serverkomponenten und zwei Client-Komponenten. Für ES/* und IS/* gibt es entsprechend jeweils $5 \cdot 3 \cdot 3 = 45$ Möglichkeiten für eine Serverkomponente und zwei Client-Komponenten. Insgesamt sind dies $225 + 225 + 45 + 45 = 540$ theoretisch denkbare Möglichkeiten.

[17] Diese Übersicht wurde aus den vorherigen Abschnitten automatisch generiert, siehe Anhang E.

4. Evaluation

Die in Kapitel 3 entworfenen 78 Architekturen, die allesamt die Anforderungen der Priorität *notwendig* erfüllen, werden nun anhand verschiedener Kriterien evaluiert. Das Ziel ist, eine möglichst geeignete Architektur für den Prototypen auszuwählen. Dabei kommen nicht nur allgemeingültige Kriterien zum Tragen, sondern vor allem auch Kriterien, die aus konkreten Anforderungen abgeleitet werden.

Wie beim Entwurf der Architekturen kommt auch hier ein sehr systematisches Verfahren zum Einsatz, was angesichts der 78 zu evaluierenden Architekturen auch gar nicht anders möglich ist. Dazu wird zunächst in Abschnitt 4.1 die verwendete formale Notation eingeführt. Danach werden in Abschnitt 4.2 alle Kriterien sowohl erklärt als auch formal beschrieben. Die Auswertung erfolgt dann in Abschnitt 4.3, wobei die Architekturen zunächst rein formal auf die aussichtsreichsten Kandidaten reduziert werden, um dann mit Hilfe einer entsprechenden Übersichtstabelle eine informierte Entscheidung zu treffen.

4.1. Notation

Die konkreten Architektur-Kandidaten mit ihrer jeweiligen Auswahl der Hauptkomponenten werden auf folgende Weise notiert:

ARCH/IS-ES-IC-EC

Dabei ist:

ARCH die Struktur der Architektur (BS-IS, BS-IC, ES, IS),
IS die Art des internen Servers (SD, SH, SM, SX, E, ∅),
ES die Art des externen Servers (SD, SH, SM, SX, E, ∅),
IC die Art des internen Clients (EN, ER, EK, ∅) und
EC die Art des externen Clients (EN, ER, EK, ∅).

Zum Beispiel wurden in Unterabschnitt 3.3.5 die folgenden Varianten der Architekturfamilie IS/* ermittelt:

Arch.	Int. Server	Ext. Server	Int. Client	Ext. Client	#	Σ
IS/*	E	∅	EN,ER,EK	ER,EK	6	
IS/*	SH	∅	EN,ER	ER	2	8

Dies entspricht der folgenden Liste von 8 Architektur-Kandidaten:

IS/E-∅-EN-ER	IS/E-∅-ER-ER	IS/E-∅-EK-ER	IS/SH-∅-EN-ER
IS/E-∅-EN-EK	IS/E-∅-ER-EK	IS/E-∅-EK-EK	IS/SH-∅-ER-ER

Die Bewertungskriterien werden mit kleinen griechischen Buchstaben bezeichnet, zum Beispiel ω. Kriterien können aus mehreren unabhängig voneinander zu untersuchenden Teil-Kriterien bestehen, diese erhalten jeweils einen entsprechenden Index, zum Beispiel ω_1 und ω_2.

Jedes Kriterium wird auf alle Architektur-Kandidaten angewendet. Das Ergebnis ist jeweils eine Zahl, die jedoch nur als Rangfolge und nicht als Metrik zu verstehen ist.[1] Auf die Beschreibung jedes Kriteriums folgt eine formale Konkretisierung, welche Architektur-Kandidaten welche Bewertung erhalten. Um Platz zu sparen, werden kommaseparierte Listen sowie der Platzhalter * verwendet.[2] Eine Zeile in der formalen Konkretisierung könnte beispielsweise so aussehen:

Krit.	Arch.	Int. Server	Ext. Server	Int. Client	Ext. Client	Bew.
ω_1	IS	S*	*	EN,ER	*	1

Entsprechend würden folgende Architektur-Kandidaten der Architekturfamilie IS/* in Bezug auf des Kriterium ω_1 die Bewertung 1 erhalten:

Architektur	ω_1
IS/SH-∅-EN-ER	1
IS/SH-∅-ER-ER	1

4.2. Kriterien

Es folgt eine Zusammenstellung aller Kriterien, anhand derer die Architektur-Kandidaten miteinander verglichen werden. Jedes Kriterium wird zunächst ausführlich erklärt und die gegebenenfalls zu Grunde liegenden Anforderungen werden konkret benannt. Danach folgt jeweils eine formale Konkretisierung in der in Abschnitt 4.1 erklärten Notation. Um die Auswertung in Abschnitt 4.3 nicht vorweg zu nehmen, werden in diesem Abschnitt alle Kriterien und Teil-Kriterien völlig unabhängig voneinander betrachtet. Insbesondere werden sie weder gewichtet noch gegeneinander aufgewogen. Auch ihre Reihenfolge ist willkürlich und hat keine Bedeutung.

α: Aufwand

Die folgenden Teil-Kriterien dienen der groben Abschätzung verschiedener Aspekte des Entwicklungsaufwandes.

Wie bereits in Abschnitt 3.3 erläutert, stellt die direkte Anwendbarkeit einer fertigen, leicht integrierbaren, gut getesteten Software einen qualitativen Vorteil dar. Dies war die Motivation dafür, überhaupt eine strikte Unterscheidung zwischen Standardkomponenten und Eigenentwicklungen vorzunehmen. Daher werden mit α_1 und α_2 die zugehörigen Bewertungskriterien für den externen und den internen Server eingeführt.

[1] Das heißt, „2“ ist besser als „1“, aber nicht unbedingt doppelt so gut. Zudem ist eine „1“ nicht unbedingt genauso viel wert wie eine „1“ bezüglich eines anderen Kriteriums.

[2] Jede Zeile kann somit auch als regulärer Ausdruck verstanden werden. Das genannte Beispiel entspricht dem Ausdruck: `^IS/(S.*)-(.*)-(EN|ER)-(.*)$`

Diese Unterscheidung ist auf die Client-Komponenten leider nicht direkt anwendbar, da für diese ohnehin nur Eigenentwicklungen in Betracht kommen. Stattdessen müsste hier der Aufwand für die *Art* der Eigenentwicklung (EN, ER, EK) abgeschätzt werden. Da es in der einschlägigen Fachliteratur jedoch keinen Hinweis auf einen prinzipiellen Unterschied im Entwicklungsaufwand für native Clients (EN), Rich Internet Applications (ER) und klassischen Webapplikationen (EK) gibt, können hier leider nur schwächere Aussagen bezüglich des Aufwandes getroffen werden. In α_3 wird die Wiederverwendbarkeit von STANLY_ACOS-Komponenten untersucht, was jedoch keine Unterscheidung zwischen EN und EK ermöglicht. In α_4 wird die Art des externen Clients mit der Art des internen Clients in Verbindung gebracht.

α_1: Aufwand: Standardkomponente versus Eigenentwicklung (externer Server)

Die Abschätzung des Aufwandes für den externen Server erfolgt auf naheliegende Weise. Die niedrigste Bewertung (hier: 0) erhalten entsprechend die Architekturen, in denen der externe Server als Eigenentwicklung realisiert wird. Eine bessere Bewertung (hier: 1) erhalten die Architekturen, die eine der Standardkomponenten als externen Server vorsehen. Die höchste Bewertung (hier: 2) erhalten die Architekturen, die ohne externen Server auskommen.

Gegen letzteres könnte man argumentieren, dass die Arbeit des externen Servers ja trotzdem getan werden muss, der Aufwand sich also lediglich auf den internen Server oder Client verschiebt. Doch selbst in diesem Fall würde zumindest die Kommunikation mit dem externen Server wegfallen. Daher wird an dieser Stelle der Wegfall des externen Servers, wenn sich sonst nichts an der Architektur ändert, auf jeden Fall als Verringerung des Entwicklungsaufwandes angesehen.

Krit.	Arch.	Int. Server	Ext. Server	Int. Client	Ext. Client	Bew.
α_1	*	*	E	*	*	0
α_1	*	*	S*	*	*	1
α_1	*	*	$\varnothing$	*	*	2

α_2: Aufwand: Standardkomponente versus Eigenentwicklung (interner Server)

Analog zu Kriterium α_1 erhalten die Architekturen mit selbstentwickeltem internen Server eine schlechtere Bewertung (1) als diejenigen, die mit einer Standardkomponente auskommen (2).

Im Gegensatz zu α_1 erhalten Architekturen ohne internen Server jedoch die schlechteste Bewertung (0), denn bei diesen handelt es sich um ES/* (Unterabschnitt 3.3.4). Das heißt, es verlagert sich nicht nur die gesamte Funktionalität des internen Servers auf die internen Clients, sondern die internen Clients müssen zudem ein stabiles, selbstorganisiertes Peer-to-Peer-Netzwerk bilden. Dies ist eine deutlich höhere Herausforderung in der Entwicklung als eine Client-Server-Architektur.

Krit.	Arch.	Int. Server	Ext. Server	Int. Client	Ext. Client	Bew.
α_2	ES	$\varnothing$	*	EN	*	0
α_2	*	E	*	*	*	1
α_2	*	S*	*	*	*	2

α_3: Aufwand: Wiederverwendbarkeit von STANLY_ACOS-Komponenten

Ist der interne Client eine Rich Internet Application (ER), so hat er die Möglichkeit, GUI-Komponenten und andere Funktionalitäten direkt von STANLY_ACOS zu verwenden, da es sich beim Client von STANLY_ACOS ebenfalls um eine Rich Internet Application handelt. Beispiele hierfür sind die Kartendarstellung und die Interaktionen mit der Karte, wie in Anforderung 18 oder auch Anforderung 47 gefordert. Weitere Synergien könnten sich in der Darstellung militärischer Luftraum-Buchungen entsprechend Anforderung 55 ergeben, auch wenn dies nur eine optionale Anforderung ist.

Krit.	Arch.	Int. Server	Ext. Server	Int. Client	Ext. Client	Bew.
α_3	*	*	*	EN,EK	*	0
α_3	*	*	*	ER	*	1

α_4: Aufwand: Synergien in der Entwicklung der Clients

Wenn der interne und der externe Client gleichartig aufgebaut sind (beide EN, beide ER oder beide EK), so sind Synergien in der Entwicklung zu erwarten. Dies gilt umso mehr, da nach Anforderung 14 nahezu alle Funktionalitäten des externen Clients auch im internen Client vorhanden sind.

Krit.	Arch.	Int. Server	Ext. Server	Int. Client	Ext. Client	Bew.
α_4	*	*	*	EN	ER,EK	0
α_4	*	*	*	ER	EN,EK	0
α_4	*	*	*	EK	EN,ER	0
α_4	*	*	*	EN	EN	1
α_4	*	*	*	ER	ER	1
α_4	*	*	*	EK	EK	1

β: Zugänglichkeit des externen Clients

In den Architekturen der Struktur IS/* (Unterabschnitt 3.3.5) müssen alle Nutzer des externen Clients auf den internen Server zugreifen. Das ist nur möglich, wenn diese Nutzer vorher am SSL-Gateway (siehe Anforderung 41) registriert werden, was mit hohem Aufwand und Kosten für die DFS verbunden ist. So könnte der externe Client letztendlich nur den Vielnutzern zur Verfügung gestellt werden. Für alle anderen Architekturen, in denen es einen externen Server gibt, stellt sich dieses Problem nicht.

Krit.	Arch.	Int. Server	Ext. Server	Int. Client	Ext. Client	Bew.
β	IS	*	$\varnothing$	*	*	0
β	BS-*,ES	*	E,S*	*	*	1

γ: Offene Schnittstelle

Dieses Kriterium bewertet, inwieweit die Architektur eine offene Schnittstelle (Anforderung 49) für alternative externe Clients unterstützt.
Gibt es keinen externen Server (siehe Unterabschnitt 3.3.5), so müsste die offene Schnittstelle durch das SSL-Gateway hindurch realisiert werden, was, wenn überhaupt, nur äußerst umständlich möglich wäre.

Gibt es einen externen Server, so würde im Falle eines nativen Clients (EN) oder einer Rich Internet Application (ER) ohnehine eine klare Schnittstelle zwischen dem externen Server und Client existieren. Daher werden diese Architekturen höher bewertet als klassische Webapplikationen, in denen solch eine Schnittstelle erst noch etabliert werden müsste.

Krit.	Arch.	Int. Server	Ext. Server	Int. Client	Ext. Client	Bew.
γ	IS	*	$\varnothing$	*	*	0
γ	*	*	E	*	EK	1
γ	*	*	E,S*	*	EN,ER	2

δ: Robustheit der Datenübertragung vom externen Server

Falls der externe Server ein XMPP-Server ist (SX), und die dort eingelieferten Vorhaben vom internen Client abgerufen werden (BS-IC/*, siehe Unterabschnitt 3.3.2), so ist die Datenübertragung weniger robust als in anderen Architekturen. Das kritische Szenario ist, dass ein Client die XMPP-Nachrichten abfragt und die abgeholten Daten nicht rechtzeitig auf den internen Server überträgt, sei es durch versehentliches Beenden oder einen Bedienungsfehler. In diesem Fall können die verlorenen Nachrichten nicht erneut vom Server abgefragt werden, da solch eine Funktionalität in XMPP nicht vorgesehen ist. Diese Architekturen erhalten daher die schlechteste Bewertung (hier: 0).

Doch auch im Falle eines anderen externen Servers (E, SH oder SM) stellt sich in den Architekturen BS-IC/* die Herausforderung, dass zwischen den internen Clients eine Koordination notwendig ist, sodass Vorhaben nicht doppelt abgeholt werden. Daher erhalten diese Architekturen nur eine etwas bessere Bewertung (hier: 1).

In den übrigen Architekturen BS-IS/*, ES/* und IS/* sind diese Probleme von vornherein ausgeschlossen, daher erhalten diese die höchste Bewertung (hier: 2).

Krit.	Arch.	Int. Server	Ext. Server	Int. Client	Ext. Client	Bew.
δ	BS-IC	*	SX	*	*	0
δ	BS-IC	*	E,SH,SM	*	*	1
δ	BS-IS,ES,IS	*	*	*	*	2

ε: Same-Origin-Policy

Wenn die auf dem externen Server eingelieferten Vorhaben vom internen Client abgerufen werden (BS-IC/*, siehe Unterabschnitt 3.3.2), stellt sich eine besondere Herausforderung, falls der interne Client eine Webapplikation ist (ER oder EK). In diesem Fall unterliegt der interne Client der Same-Origin-Policy, das heißt, er darf nur Daten vom internen Server nachladen, da er über diesen ausgeliefert wurde. AJAX-Aufrufe zum externen Server werden vom Browser untersagt.

Die klassische Lösung für dieses Problem ist JSONP [Ipp05], das heißt, der Client lädt JavaScript-Code vom externen Server nach[3] und vertraut darauf, dass dieser Code eine gegebene Callback-Funktion mit den gewünschten Daten als Parameter aufruft, und sonst nichts weiter tut. Aus Sicherheitsgründen und auch wegen

[3] Dies ist kurioserweise erlaubt und wird durch die Same-Origin-Policy nicht verhindert.

Anforderung 44 wird dieses Verfahren hier nicht weiter betrachtet, da es dem externen Server erlaubt, einen beliebigen Programmcode in die internen Clients zu injizieren.
Die moderne Lösung für dieses Problem ist CORS [Wor14], ein relativ junger Standard, der vom Browser unterstützt werden muss und serverseitig die Behandlung spezieller HTTP-Header voraussetzt.
Da CORS derzeit noch nicht von allen Standard-HTTP-Komponenten unterstützt wird und zudem im Internet Explorer 9 (Anforderung 43) nur über eine proprietäre JavaScript-API (XDomainRequest) erreichbar ist, werden alle Architekturen höher bewertet (1), in denen diese Problematik nicht auftritt.

Krit.	Arch.	Int. Server	Ext. Server	Int. Client	Ext. Client	Bew.
ε	BS-IC	*	*	ER,EK	*	0
ε	BS-IC	*	*	EN	*	1
ε	BS-IS,ES,IS	*	*	*	*	1

ζ: Standardisierte Autorisations-Mechanismen

Wenn die auf dem externen Server eingelieferten Vorhaben abgerufen werden, wird dafür nach Anforderung 44 ein entsprechender Authentifikations- und Autorisationsmechanismus benötigt. Dieses Kriterium beschreibt, inwiefern dies von den gegebenenfalls verwendeten Standardkomponenten direkt und auf standardisierte Weise unterstützt wird.
Ist der externe Server ein Mailserver (SM), so können die E-Mails intern via POP oder IMAP abgeholt werden, wobei die Autorisation einfach darin besteht, dass ein Empfänger nur seine eigenen Nachrichten abrufen kann. Bei einem XMPP-Server (SX) sieht es genauso positiv aus, auch hier gibt es direkt einen kennwortgeschützten Account für den Empfänger der XMPP-Nachrichten.
Unbefriedigend ist die Situation hingegen bei via HTTP erreichbaren dokumentenorientierten Datenbanken (SH). Dort ist die Authentifikation zwar durch HTTP-Auth standardisiert, doch es fehlt an geeigneten Autorisations-Mechanismen. Das Einfügen muss jedem möglich sein, aber das Ändern, Löschen und Auslesen muss geschützt werden. Diese Art von Autorisation erfordert jeweils Nicht-Standard-Erweiterungen und kann zudem nicht einfach durch einen vorgeschalteten HTTP-Server nachträglich hinzugefügt werden, da dieser die erlaubten und verbotenen Requests nicht ohne weiteres voneinander unterscheiden kann.
Ist kein externer Server vorhanden ($\varnothing$), so stellt sich dieses Problem nicht und die Bewertung ist daher positiv. Ebenfalls positiv bewertet werden Eigenentwicklungen (E), da bei diesen alle Möglichkeiten offen stehen.

Krit.	Arch.	Int. Server	Ext. Server	Int. Client	Ext. Client	Bew.
ζ	*	*	SH	*	*	0
ζ	*	*	E,SM,SX,$\varnothing$	*	*	1

4.3. Auswertung

Zur Auswertung wird zunächst jedes Kriterium auf alle 78 in Abschnitt 3.4 genannten Architektur-Kandidaten angewendet. Das Ergebnis zeigt die nachfolgende Tabelle.[4]

Architektur	α_1	α_2	α_3	α_4	β	γ	δ	ε	ζ	Max
BS-IS/E-E-EN-EN	0	1	0	1	1	2	2	1	1	—
BS-IS/E-E-EN-ER	0	1	0	0	1	2	2	1	1	—
BS-IS/E-E-EN-EK	0	1	0	0	1	1	2	1	1	—
BS-IS/E-E-ER-EN	0	1	1	0	1	2	2	1	1	—
BS-IS/E-E-ER-ER	0	1	1	1	1	2	2	1	1	—
BS-IS/E-E-ER-EK	0	1	1	0	1	1	2	1	1	—
BS-IS/E-E-EK-EN	0	1	0	0	1	2	2	1	1	—
BS-IS/E-E-EK-ER	0	1	0	0	1	2	2	1	1	—
BS-IS/E-E-EK-EK	0	1	0	1	1	1	2	1	1	—
BS-IS/E-SH-EN-EN	1	1	0	1	1	2	2	1	0	—
BS-IS/E-SH-EN-ER	1	1	0	0	1	2	2	1	0	—
BS-IS/E-SH-ER-EN	1	1	1	0	1	2	2	1	0	—
BS-IS/E-SH-ER-ER	1	1	1	1	1	2	2	1	0	—
BS-IS/E-SH-EK-EN	1	1	0	0	1	2	2	1	0	—
BS-IS/E-SH-EK-ER	1	1	0	0	1	2	2	1	0	—
BS-IS/E-SX-EN-EN	1	1	0	1	1	2	2	1	1	—
BS-IS/E-SX-EN-ER	1	1	0	0	1	2	2	1	1	—
BS-IS/E-SX-ER-EN	1	1	1	0	1	2	2	1	1	—
BS-IS/E-SX-ER-ER	1	1	1	1	1	2	2	1	1	✓
BS-IS/E-SX-EK-EN	1	1	0	0	1	2	2	1	1	—
BS-IS/E-SX-EK-ER	1	1	0	0	1	2	2	1	1	—
BS-IS/E-SM-EN-EN	1	1	0	1	1	2	2	1	1	—
BS-IS/E-SM-ER-EN	1	1	1	0	1	2	2	1	1	—
BS-IS/E-SM-EK-EN	1	1	0	0	1	2	2	1	1	—
BS-IC/E-E-EN-EN	0	1	0	1	1	2	1	1	1	—
BS-IC/E-E-EN-ER	0	1	0	0	1	2	1	1	1	—
BS-IC/E-E-EN-EK	0	1	0	0	1	1	1	1	1	—
BS-IC/E-E-ER-EN	0	1	1	0	1	2	1	0	1	—
BS-IC/E-E-ER-ER	0	1	1	1	1	2	1	0	1	—
BS-IC/E-E-ER-EK	0	1	1	0	1	1	1	0	1	—
BS-IC/SH-E-EN-EN	0	2	0	1	1	2	1	1	1	—
BS-IC/SH-E-EN-ER	0	2	0	0	1	2	1	1	1	—
BS-IC/SH-E-EN-EK	0	2	0	0	1	1	1	1	1	—
BS-IC/SH-E-ER-EN	0	2	1	0	1	2	1	0	1	—
BS-IC/SH-E-ER-ER	0	2	1	1	1	2	1	0	1	✓
BS-IC/SH-E-ER-EK	0	2	1	0	1	1	1	0	1	—

[4] Diese Übersicht wurde aus den Tabellen aus Abschnitt 3.4 und Abschnitt 4.2 automatisch generiert, siehe Anhang E.

Architektur	α_1	α_2	α_3	α_4	β	γ	δ	ε	ζ	Max
BS-IC/SD-E-EN-EN	0	2	0	1	1	2	1	1	1	—
BS-IC/SD-E-EN-ER	0	2	0	0	1	2	1	1	1	—
BS-IC/SD-E-EN-EK	0	2	0	0	1	1	1	1	1	—
BS-IC/E-SH-EN-EN	1	1	0	1	1	2	1	1	0	—
BS-IC/E-SH-EN-ER	1	1	0	0	1	2	1	1	0	—
BS-IC/E-SH-ER-EN	1	1	1	0	1	2	1	0	0	—
BS-IC/E-SH-ER-ER	1	1	1	1	1	2	1	0	0	—
BS-IC/E-SX-EN-EN	1	1	0	1	1	2	0	1	1	—
BS-IC/E-SX-EN-ER	1	1	0	0	1	2	0	1	1	—
BS-IC/E-SX-ER-EN	1	1	1	0	1	2	0	0	1	—
BS-IC/E-SX-ER-ER	1	1	1	1	1	2	0	0	1	—
BS-IC/SH-SH-EN-EN	1	2	0	1	1	2	1	1	0	—
BS-IC/SH-SH-EN-ER	1	2	0	0	1	2	1	1	0	—
BS-IC/SH-SH-ER-EN	1	2	1	0	1	2	1	0	0	—
BS-IC/SH-SH-ER-ER	1	2	1	1	1	2	1	0	0	✓
BS-IC/SH-SX-EN-EN	1	2	0	1	1	2	0	1	1	—
BS-IC/SH-SX-EN-ER	1	2	0	0	1	2	0	1	1	—
BS-IC/SH-SX-ER-EN	1	2	1	0	1	2	0	0	1	—
BS-IC/SH-SX-ER-ER	1	2	1	1	1	2	0	0	1	✓
BS-IC/SD-SH-EN-EN	1	2	0	1	1	2	1	1	0	—
BS-IC/SD-SH-EN-ER	1	2	0	0	1	2	1	1	0	—
BS-IC/SD-SX-EN-EN	1	2	0	1	1	2	0	1	1	—
BS-IC/SD-SX-EN-ER	1	2	0	0	1	2	0	1	1	—
BS-IC/E-SM-EN-EN	1	1	0	1	1	2	1	1	1	—
BS-IC/SD-SM-EN-EN	1	2	0	1	1	2	1	1	1	✓
BS-IC/SH-SM-EN-EN	1	2	0	1	1	2	1	1	1	✓
ES/∅-E-EN-EN	0	0	0	1	1	2	2	1	1	—
ES/∅-E-EN-ER	0	0	0	0	1	2	2	1	1	—
ES/∅-E-EN-EK	0	0	0	0	1	1	2	1	1	—
ES/∅-SH-EN-EN	1	0	0	1	1	2	2	1	0	—
ES/∅-SH-EN-ER	1	0	0	0	1	2	2	1	0	—
ES/∅-SX-EN-EN	1	0	0	1	1	2	2	1	1	—
ES/∅-SX-EN-ER	1	0	0	0	1	2	2	1	1	—
ES/∅-SM-EN-EN	1	0	0	1	1	2	2	1	1	—
IS/E-∅-EN-ER	2	1	0	0	0	0	2	1	1	—
IS/E-∅-EN-EK	2	1	0	0	0	0	2	1	1	—
IS/E-∅-ER-ER	2	1	1	1	0	0	2	1	1	—
IS/E-∅-ER-EK	2	1	1	0	0	0	2	1	1	—
IS/E-∅-EK-ER	2	1	0	0	0	0	2	1	1	—
IS/E-∅-EK-EK	2	1	0	1	0	0	2	1	1	—
IS/SH-∅-EN-ER	2	2	0	0	0	0	2	1	1	—
IS/SH-∅-ER-ER	2	2	1	1	0	0	2	1	1	✓

Dabei gibt die letzte Spalte *Max* an, ob die Architektur ein (lokales) Maximum ist. Die Maxima werden bezüglich der natürlichen Halbordnung $\leq^n$ (*komponentenweise-kleiner-oder-gleich*) auf den Bewertungs-Tupeln ermittelt. In diesem Sinne ist eine Architektur nur dann besser-oder-gleich einer anderen, wenn sie in allen Kriterien eine bessere oder gleich gute Bewertung hat. Zwei Architekturen, die unterschiedlich gelagerte Vor- und Nachteile haben, sind in der Halbordnung $\leq^n$ nicht miteinander vergleichbar.

Diese Herangehensweise hat den Vorteil, dass sie unabhängig davon funktioniert, wie man die Kriterien gewichten möchte. In jeder beliebigen Gewichtung wird ein Nicht-Maximum stets schlechter als eines der Maxima abschneiden. Es genügt also, nur die Maxima zu betrachten. Die folgenden Tabelle fasst diese zusammen, wobei zur besseren Übersicht alle optimal erfüllten Bewertungen durch „—“ ersetzt werden. So ist leichter erkennbar, bezüglich welcher Kriterien die jeweilige Architektur nicht optimal sind:

Architektur	α_1	α_2	α_3	α_4	β	γ	δ	ε	ζ	Max
BS-IS/E-SX-ER-ER	1	1	—	—	—	—	—	—	—	✓
BS-IC/SH-E-ER-ER	0	—	—	—	—	—	1	0	—	✓
BS-IC/SH-SH-ER-ER	1	—	—	—	—	—	1	0	0	✓
BS-IC/SH-SX-ER-ER	1	—	—	—	—	—	0	0	—	✓
BS-IC/SD-SM-EN-EN	1	—	0	—	—	—	1	—	—	✓
BS-IC/SH-SM-EN-EN	1	—	0	—	—	—	1	—	—	✓
IS/SH-∅-ER-ER	—	—	—	—	0	0	—	—	—	✓

Auf Basis dieser Informationen muss nun eine Entscheidung getroffen werden.

Interessant ist die Architektur IS/SH-∅-ER-ER, da diese alle α-Kriterien optimal erfüllt und somit voraussichtlich den geringsten Entwicklungsaufwand benötigen wird. Doch dies geschieht zu einem hohen Preis: Die Kriterien β und γ zeigen die niedrigste Bewertung, das heißt, die Nutzbarkeit des externen Clients ist extrem eingeschränkt und die offene Schnittstelle ist nahezu unbrauchbar.

Es erscheint sinnvoller, genau umgekehrt zu argumentieren: Erfreulicherweise erfüllt eine der Architekturen alle Kriterien β, γ, δ, ε und ζ optimal, auch wenn dies einen etwas höheren Entwicklungsaufwand in den Serverkomponenten (α_1 und α_2) bedeutet. Das ist ein ziemlich guter Kompromiss. Die Entscheidung fällt somit auf die folgende Architektur:

BS-IS/E-SX-ER-ER

5. Prototypische Umsetzung

Die im Kapitel 4 ausgewählte Architektur BS-IS/E-SX-ER-ER wird in Form eines Prototypen umgesetzt. Das heißt:

BS Es gibt genau einen internen und einen externen Server.
IS Nur der interne Server holt die Daten vom externen Server ab.
E Der interne Server ist eine Eigenentwicklung.
SX Der externe Server ist eine Standardkomponente, genauer ein XMPP-Server.
ER Der interne Client ist eine Eigenentwicklung als Rich Internet Application.
ER Auch der externe Client ist eine Eigenentwicklung als Rich Internet Application.

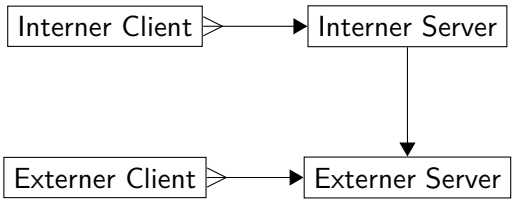


Abbildung 17.: Architektur BS-IS/*. Siehe Unterabschnitt 3.3.1.

Der Prototyp soll in erster Linie demonstrieren, dass diese Architektur wirklich funktioniert. Darüber hinaus soll er als Diskussions-Grundlage für zukünftige Entwicklungen dienen, die von nun an mit den Nutzern konkret anhand des Prototypen abgesprochen werden können.

Der Prototyp erfüllt naturgemäß nur einen Teil der Anforderungen, was gleich zu Beginn in Abschnitt 5.1 klargestellt wird. Danach werden in Abschnitt 5.2 bis Abschnitt 5.16 alle wesentlichen Aspekte des Prototypen beschrieben. Während der Entwicklung stellten sich einige besondere Herausforderungen, die in Anhang D erläutert werden.

5.1. Bewusst gewählte Einschränkungen

Da die Anforderungen sehr umfangreich sind und den Rahmen einer Diplomarbeit bei weitem übersteigen, wird im Prototyp nur ein Teil der Anforderungen umgesetzt. Dabei wurden vor allem solche Anforderungen fallen gelassen, die in Bezug auf die Architektur keine besondere Herausforderung darstellen, und zudem unwesentlich für eine erste Präsentation gegenüber den Nutzern sind:

Keine Tests im Internet Explorer

Im Gegensatz zu Anforderung 43 hat der Prototyp keine gründlichen Tests im Internet Explorer erfahren, da auf allen Büro-Rechnern der DFS neuerdings ein aktueller Firefox in der Version ≥ 20 läuft.[1]

Keine Lufträme aus Grunddaten

Im Gegensatz zu Anforderung 28 können die Luftraum-Grunddaten weder importiert noch in den Vorhaben verwendet werden. Es ist nur die in Anforderung 8 beschriebene direkte Koordinateneingabe möglich. Der Umgang mit festen Luftraum-Strukturen aus Grunddaten ist bereits in STANLY_ACOS möglich und braucht daher durch den Prototyp nicht validiert zu werden. Da die konzeptuelle Neuerung im BNL-Modul gerade die Möglichkeit von ad hoc definierten Lufträumen ist, liegt der Fokus erst einmal darauf.

Keine Speicherung der Geometrie-Quelldaten

Die entsprechend Anforderung 8 eingegebenen Geometrien werden nur in der für die Karten-Darstellung benötigten Form gespeichert. Zum Beispiel wird von einem Kreis weder Mittelpunkt noch Radius gespeichert, sondern nur das interpolierte Polygon (siehe Abschnitt 5.9). Eingeladene OVL-Dateien und andere Texteingaben werden entsprechend ebenfalls nicht im Original gespeichert, sondern nur die daraus extrahierte Geometrie.

Irrelevante Eingabefelder

Für jede Vorhabens-Art werden sämtliche Eingabefelder angeboten, egal ob sie laut Anforderung 1 benötigt werden oder nicht.

Nur ein Zeitbereich und keine Teilvorhaben

Für jedes Vorhaben kann nur ein einziger Zeitbereich eingegeben werden, statt mehrerer Zeitbereiche (Anforderung 4). Zudem gibt es keine direkte Unterstützung für mehrteilige BNL-Vorhaben (Anforderung 5). Als Workaround wurde ein Copy-Button eingeführt, mit dem bestehende Vorhaben kopiert werden können. Mehr dazu in Abschnitt 5.4.

Uhrzeiten in UTC

Im Gegensatz zu Anforderung 3 erfolgt die Eingabe der Uhrzeiten in UTC statt Lokalzeit, denn bisher arbeitet STANLY_ACOS vollständig auf Basis von UTC.

Ineffiziente automatische Aktualisierung

Die automatische Aktualisierung lädt bei jeder kleinen Änderung alles neu, was eine hohe Sicherheit garantiert, aber in einer produktiv genutzten Anwendung optimiert werden sollte. Mehr dazu in Abschnitt 5.7.

Parallele Bearbeitung

Wenn ein Vorhaben von mehreren Benutzern gleichzeitig bearbeitet wird, überschreibt die zuletzt gespeicherte Änderung alle vorherigen Änderungen. Das sollte verbessert werden. Es gibt jedoch noch keine Anforderung, die ein Konzept zur Konfliktbehandlung bei paralleler Bearbeitung beschreibt.[2]

[1] Ungeachtet dessen ist der Internet Explorer weiterhin Unternehmensstandard in der DFS für Neuentwicklungen. Das heißt, ein zukünftiger Entwicklungsauftrag wird nach wie vor auf Anforderung 43 bestehen.

[2] Absehbar ist, dass hier ein Locking-Mechanismus sinnvoll wäre. Das heißt, wenn ein Nutzer ein Vor-

Keine automatische Aktualisierung während der Bearbeitung
: Während der Benutzer ein Vorhaben bearbeitet, führt der Client automatische Aktualisierungen nicht sofort durch, sondern erst nach Abschluss der Bearbeitung. Mehr dazu in Abschnitt 5.10.

Keine Anzeige militärischer Luftraum-Buchungen
: Im Gegensatz zu Anforderung 55 werden in der Karte nur die BNL-Vorhaben angezeigt und keine militärischen Luftraum-Buchungen.

Supervisorreminder in Formular statt Liste
: Im Gegensatz zu Anforderung 48 befinden sich die Checkboxen zum Setzen der Supervisorreminder im Formular und nicht direkt in der Vorhabensliste, siehe Abschnitt 5.8.

Keine Selektierung von Vorhaben über die Karte
: Im Gegensatz zu Anforderung 18 können Vorhaben nicht durch einen Klick auf die Karte ausgewählt werden, sondern nur direkt in der Vorhabens-Liste. Mehr dazu in Abschnitt 5.8.

Kein externer Server und externer Client
: Im Gegensatz zu Anforderung 14 und Anforderung 49 wird auf Umsetzung des externen Servers und des externen Clients verzichtet. Der externe Server ist ohnehin eine Standardkomponente (XMPP-Server) und der externe Client ist im Wesentlichen eine reduzierte Version des internen Clients. Auch die Kommunikation einer Rich Internet Application mit einem XMPP-Server erfordert keine gesonderte Untersuchung, da STANLY_ACOS bereits demonstriert, dass dies ohne weiteres möglich ist (siehe Abbildung 7).

5.2. Verortung des Prototypen

Der Architektur entsprechend ist der BNL-Prototyp nicht als separate Software, sondern als Modul von STANLY_ACOS realisiert, das sich nahtlos in das bereits vorhandene Framework eingliedert. Der prinzipielle Aufbau der STANLY_ACOS-Module wurde bereits in Abbildung 8 und Abbildung 9 im Abschnitt 3.2 erläutert.

Doch auch STANLY_ACOS existiert nicht in Isolation, sondern baut auf verschiedenen Softwarepaketen auf, die im umgebenden System zur Verfügung stehen müssen. Mehr dazu in Anhang C.

Der Prototyp basiert auf dem letzten Release von STANLY_ACOS im Jahr 2013 im Entwicklungs-Branch *master*. [DFS13b] Von dieser Version ausgehend gibt es einen neuen Entwicklungs-Branch *bnl* für den BNL-Prototypen.

---

haben editiert, wird dieses für alle anderen Nutzer gesperrt. Nach einer gewissen Wartezeit sollte die Sperre automatisch aufgehoben werden. Dann kann ein anderer Nutzer die Bearbeitung beginnen, woraufhin der ursprüngliche Nutzer nichts mehr speichern kann. Möglicherweise ist zusätzlich auch eine Historie sinnvoll, in der genau aufgezeichnet wird, welcher Nutzer zu welchem Zeitpunkt welche Änderung vorgenommen hat.

5.3. Dateien-Übersicht

Es folgt eine Übersicht über alle Dateien und Ordner von STANLY_ACOS, die für diese Arbeit relevant sind. Die **hervorgehobenen** Dateien und Ordner sind für den Prototypen neu hinzugekommen. Die übrigen existierten bereits in STANLY_ACOS und wurden für den Prototypen lediglich angepasst. Für jede Datei ist ausgeführt, in welchem der folgenden Abschnitte sie behandelt wird. Einige Dateien werden in mehreren Abschnitten behandelt.

- client/App/
  - App.js ..... Abschnitt 5.16
  - **module/bnl/**
    - **controller/**
      - **Bnl.js** ..... Abschnitt 5.7, Abschnitt 5.10
    - **model/**
      - **Bnl.js** ..... Abschnitt 5.5
      - **Bnlcenter.js** ..... Abschnitt 5.5
      - **Bnltype.js** ..... Abschnitt 5.5
      - **UomHorizontal.js** ..... Abschnitt 5.5
      - **UomVertical.js** ..... Abschnitt 5.5
    - **store/**
      - **Bnl.js** ..... Abschnitt 5.5
      - **Bnlcenter.js** ..... Abschnitt 5.5
      - **Bnltype.js** ..... Abschnitt 5.5
      - **UomHorizontal.js** ..... Abschnitt 5.5
      - **UomVertical.js** ..... Abschnitt 5.5
    - **view/**
      - **Bnl.js** ..... Abschnitt 5.8
      - **BnlForm.js** ..... Abschnitt 5.8
      - **BnlGrid.js** ..... Abschnitt 5.8
      - **BnlContextMenu.js** ..... Abschnitt 5.8
  - module/main/
    - view/
      - Menu.js ..... Abschnitt 5.16
  - util/
    - **Container.js** ..... Abschnitt 5.10
    - **GeometryParser.js** ..... Abschnitt 5.13
  - widget/
    - **DateTimeFieldBnl.js** ..... Abschnitt 5.12
    - **GeometryField.js** ..... Abschnitt 5.13

- client/build/
  - Ext_Loader_history.txt ..... Abschnitt 5.16
- server/app/
  - Controller/
    - **BnlController.php** ..... Abschnitt 5.7, Abschnitt 5.6
  - Model/
    - **Crud.php** ..... Abschnitt 5.14
- server/db/
  - upgrade.sql ..... Abschnitt 5.15
  - init_demo.sql ..... Abschnitt 5.15
  - auth/
    - table_fix/
      - permission.sql ..... Abschnitt 5.15
      - role.sql ..... Abschnitt 5.15
      - role_permission.sql ..... Abschnitt 5.15
  - **bnl/**
    - **upgrade_module.sql** ..... Abschnitt 5.15
    - **func/**
      - **interpolate_circle.sql** ..... Abschnitt 5.9
    - **table_fix/**
      - **bnlcenter.sql** ..... Abschnitt 5.4
      - **bnltype.sql** ..... Abschnitt 5.4
      - **uom_horizontal.sql** ..... Abschnitt 5.4
      - **uom_vertical.sql** ..... Abschnitt 5.4
    - **table_live/**
      - **bnl.sql** ..... Abschnitt 5.4

5.4. Datenmodell

Das Datenmodell wird in der Datenbank über meherere Relationen aufgebaut, die in den folgenden Dateien definiert sind:

- server/db/
  - bnl/
    - table_fix/
      - bnlcenter.sql
      - bnltype.sql
      - uom_horizontal.sql
      - uom_vertical.sql
    - table_live/
      - bnl.sql

Da im Prototyp fast ausschließlich CRUD-Operationen[3] zum Einsatz kommen, gibt es nur Tabellen und keine Views oder Service-Funktionen (in Abschnitt 3.2, Abbildung 8 aufgeführt). Die Tabellen entsprechen 1:1 den in Abschnitt 5.5 beschriebenen Models im Client.

In *table_fix* liegen Stammdaten-Tabellen, deren SQL-Datei jeweils sowohl die Tabellen-Definitionen als auch die Inhalte enthält. In *table_live* hingegen liegen die Tabellen, deren Inhalt sich ständig im laufenden Betrieb ändert. Diese haben jeweils nur eine Definition und keinen vordefinierten Inhalt.[4]

Der Prototyp hat nur eine veränderbare Tabelle *bnl* in *table_live*. Diese speichert die BNL-Vorhaben und zeigt über verschiedene Fremdschlüssel in die Stammdaten-Tabellen. Fremdschlüsselbeziehungen zwischen den Stammdaten-Tabellen gibt es nicht, da es auch keine inhaltlichen Zusammenhänge zwischen diesen gibt.

Die Stammdaten-Tabelle *bnlcenter* beschreibt die in Anforderung 2 aufgeführten möglichen Center, denen ein BNL-Vorhaben zugewiesen sein kann. Der Namenspräfix *bnl* ist notwendig, um eine Namenskollision mit der bereits in STANLY_ACOS vorhandenen Tabelle *center* zu verhindern. Dabei ist *bnlcenter* eine Teilmenge von *center*, was durch einen entsprechenden Fremdschlüssel explizit sichergestellt wird. Zudem enthält *bnlcenter* eine zusätzliche Spalte *center_label*, da im BNL-Prototyp geringfügig andere Center-Bezeichnungen als im Rest von STANLY_ACOS verwendet werden sollten.[5]

Die Stammdaten-Tabelle *bnltype* listet die in Anforderung 1 beschriebenen Vorhabens-Typen auf. Der Namenspräfix *bnl* ist notwendig, weil *type* ein viel zu allgemeiner Begriff für einen Tabellennamen ist. Zudem ist *type* ein Schlüsselwort in SQL. [The14, S. 1882]

Die Stammdaten-Tabelle *uom_horizontal* enthält die horizontalen Maßeinheiten für Abstände auf dem Erdboden, wie in Anforderung 45 aufgeführt. Entsprechend beinhaltet *uom_vertical* die vertikalen Maßeinheiten für Höhenangaben nach Anforderung 1. Der Namenspräfix *uom* steht jeweils für *unit of measurement*.

Die Tabelle *bnl* enthält die eigentlichen BNL-Vorhaben und entspricht genau dem, was in der Vorhabensliste im Client zu sehen ist (siehe Abschnitt 5.8). Sie enthält genau die Felder der Eingabemaske aus Anforderung 1, mit folgenden Abweichungen:

1. Das in Anforderung 2 beschriebene Feld *Unique ID* wird nicht durch eine Spalte, sondern durch mehrere Spalten kodiert, da die Einzelteile der *Unique ID* mehrere unabhängige Informationen darstellen: das verwaltende Center (*center*), die fortlaufende Zahl (*bnlnumber*) und das zugeordnete Jahr (*year*). Diese Darstellung erleichtert die interne Handhabung des Modells, und hat zudem den Vorteil, dass

[3] Datensätze erzeugen, einlesen, verändern und löschen (Create, Read, Update, Delete)

[4] Darüber hinaus gibt es in STANLY_ACOS noch den dritten Tabellentyp *table_basicdata*, der Stammdaten-Tabellen enthält, die über die Benutzeroberfläche vom Administrator verwaltet werden können. Diese sind im Prinzip *table_live*-Tabellen, müssen jedoch an verschiedenen Stellen administrativ anders behandelt werden, etwa beim Übertragen von getesteten Stammdaten vom Testsystem ins Livesystem. Im BNL-Prototyp kommen keine *table_basicdata*-Tabellen zum Einsatz, da eine entsprechende Administrationsoberfläche den Rahmen dieser Diplomarbeit sprengen würde.

[5] Da *center* auch einige Center der Nachbarländer enthält, werden dort die vollständigen Bezeichner inklusive Landeskenner (zum Beispiel *EDGG*) verwendet. Im BNL-Prototyp hingegen sollen die kurzen Bezeichner (zum Beispiel *GG*) ohne Landeskennung verwendet werden.

das Jahr komplett erfasst wird und nicht nur in Form seiner letzten beiden Ziffern. Hierdurch wird das BNL-Modul Y2.1K-sicher[6].
2. Der in Anforderung 2 beschriebene optionale Anhang für Teilvorhaben (fortlaufender Kleinbuchstabe) wurde vorerst weggelassen. Es gibt im Prototypen keine direkte Unterstützung für mehrteilige BNL-Vorhaben. Mehr dazu weiter unten.
3. Für den in Anforderung 48 beschriebenen Supervisorreminder wurden mehrere Boolean-Felder nach dem Namensschema *reported_** ergänzt.
4. Es musste eine künstliche ID-Spalte *bnl_id* eingeführt werden, obwohl diese vom Datenmodell her eigentlich überflüssig ist. Mehr dazu in Anhang D.

Die Tabelle *bnl* zeigt mit folgenden Fremdschlüsseln in die Stammdaten-Tabellen hinein:

Feldname	Spalte in Tabelle *bnl*	Stammdaten-Tabelle (Schlüssel)
Unique ID / Center	center	bnlcenter (center)
Type	bnltype	bnltype (bnltype)
Min height / UOM	lower_uom	uom_vertical (uom_vertical)
Max height / UOM	upper_uom	uom_vertical (uom_vertical)
Separation buffer	sep_uom	uom_horizontal (uom_horizontal)

Diese Fremdschlüssel entsprechen genau den Auswahllisten des in Abschnitt 5.8 beschriebenen Eingabeformulars.

Nach Anforderung 5 soll ein BNL-Vorhaben aus mehreren Teilen bestehen können. Das heißt, die Tabelle *bnl* müsste eigentlich in zwei Tabellen zerlegt werden, wobei eine Tabelle die Informationen beinhaltet, die sich von Teilvorhaben zu Teilvorhaben nicht ändern. Dann wäre das Datenmodell in Boyce-Codd-Normalform (BCNF).

Jedoch ist eine solche Aufspaltung nur dann sinnvoll, wenn sie auch vom Client dargestellt und editiert werden kann. Dies hätte viele weitere Ansprüche an die Benutzeroberfläche nach sich gezogen und damit den zeitlichen Rahmen dieser Diplomarbeit gesprengt. Daher wurde auf diese Aufspaltung verzichtet. Als Workaround können vorhadene Vorhaben kopiert und dann abgeändert werden, sodass der Nutzer auf diese Weise mehrteilige Vorhaben ohne erhöhten Tippaufwand eintragen kann. Die dadurch möglichen Inkonsistenzen müssen durch Aufmerksamkeit des Nutzers vermieden werden, was für diesen Prototypen aber vollkommen ausreicht, da dieser Aspekt nichts mit der Tragfähigkeit der gewählten Architektur zu tun hat, sondern in erster Linie eine aufwändige Verbesserung der Benutzeroberfläche darstellt. Durch diese fehlende Aufspaltung liegt das aktuelle Datenmodell aber nicht in BCNF vor, sondern nur in erster Normalform (1NF).

5.5. Client-Models und Synchronisation mit Server

Die BNL-Vorhaben sowie die Inhalte der Auswahlfelder (Center, Vorhabens-Typen, …) werden im Client durch entsprechende Model-Klassen repräsentiert. Weiterhin kommen ExtJS-spezifische *Store*-Klassen zum Einsatz, die weiter unten erklärt werden. Die

[6] Jahr-2100-sicher, siehe auch Y2K-Bug / Jahr-2000-Problem [Wes13, 3:11:40–3:30:32]

Model- und Store-Klassen befinden sich in folgenden Dateien:[7]

- client/App/
  - module/bnl/
    - model/
      - Bnl.js
      - Bnlcenter.js
      - Bnltype.js
      - UomHorizontal.js
      - UomVertical.js
    - store/
      - Bnl.js
      - Bnlcenter.js
      - Bnltype.js
      - UomHorizontal.js
      - UomVertical.js

Der Client verwendet im Wesentlichen das gleiche Datenmodell wie der Server. Daher entspricht jede Datenbank-Relation aus Abschnitt 5.4 einer Model-Klasse:

DB-Tabelle	Model-Klasse
bnl	App.module.bnl.model.Bnl
bnlcenter	App.module.bnl.model.Bnlcenter
bnltype	App.module.bnl.model.Bnltype
uom_horizontal	App.module.bnl.model.UomHorizontal
uom_vertical	App.module.bnl.model.UomVertical

Im Folgenden wird der Modul-Namespace *App.module.bnl* weggelassen.

Das ExtJS-Framework bietet durch sogenannte *Associations* die Möglichkeit, Fremdschlüssel-Beziehungen ebenfalls in den Model-Klassen auszudrücken. Von diesem Feature macht der Prototyp jedoch keinen Gebrauch, da die durch Associations ermöglichten Funktionalitäten (beispielsweise das Laden von verschachtelten Model-Daten vom Server) nicht benötigt werden. Zudem ist diese Funktionalität relativ neu und wirkt sich negativ auf die clientseitige Performance aus. [Sen13b]

Alle Model-Klassen sind von *Ext.data.Model* abgeleitet und erfüllen insbesondere die vom ExtJS-Framework geforderte Schnittstelle für Models. Diese besteht vor allem aus generischen Methoden zur Abfrage, welche Datenfelder vorhanden sind, sowie generischen Methoden zum Auslesen und Setzen dieser Felder. Dies ist an vielen Stellen nützlich, etwa in der tabellarischen Darstellung aller BNL-Vorhaben oder beim Befüllen und Auslesen der Vorhabensdaten aus dem Formular. Mehr dazu in Abschnitt 5.10.

[7] Es gilt die in JavaScript übliche Konvention, dass Klassennamen in großgeschriebenem CamelCase notiert werden, dass sich jede Klasse in einer eigenen Datei befindet und dass die Ordnerstruktur den Namensräumen entspricht.

Zudem ermöglicht es diese Schnittstelle, Model-Instanzen in *Stores* zu verwalten. Ein *Store* im ExtJS-Framework ist ein intelligentes Array, das heißt, eine homogene[8] Liste von Model-Instanzen mit zusätzlichen Methoden und Anbindung an das Event-System von ExtJS. So können sich andere Komponenten an einen Store andocken, um über Aktionen in diesem Store (Änderungen, Löschungen, …) benachrichtigt zu werden. Wird ein Store gefiltert, umsortiert oder ähnliches, wirkt sich dies sofort auf alle View-Komponenten aus, die den Inhalt dieses Stores anzeigen. Darüber hinaus kann ein Store seinen Inhalt mit dem Server synchron halten, und unterstützt dabei serverseitige Filterung, Sortierung und Paging[9], indem entsprechende Parameter bei der Abfrage automatisch mit zum Server gesendet werden.[10] Im Gegensatz zum Model enthält ein Store praktisch keine clientseitige Logik, sondern wird im Wesentlichen nur auf die serverseitigen Besonderheiten hin konfiguriert. Dennoch sind die Stores nicht einfach Instanzen von *Ext.data.Store*, sondern Singleton-Unterklassen davon.[11]

Es folgt eine Übersicht über alle im Prototyp definierten Stores, welche serverseitigen CRUD-Operationen sie auslösen und was für Models sie beinhalten:

Store	Create	Read	Update	Delete	Inhalt des Stores
store.Bnl	✓	✓	✓	✓	model.Bnl
store.Bnlcenter	—	✓	—	—	model.Bnlcenter
store.Bnltype	—	✓	—	—	model.Bnltype
store.UomHorizontal	—	✓	—	—	model.UomHorizontal
store.UomVertical	—	✓	—	—	model.UomVertical

Im Prototypen gibt es somit für jedes Model jeweils nur einen einzigen Store. Das ist deshalb möglich, weil ein und der selbe Store an mehreren Stellen verwendet werden kann. Zum Beispiel wird *store.UomVertical* für zwei Auswahllisten verwendet – die Maßeinheiten der minimalen Höhe und die der maximalen Höhe. Auch die Vorhabensliste *store.Bnl* wird letztlich nur einmal benötigt. Anders sähe es aus, wenn beispielsweise unterschiedliche Filterungen der Vorhabensliste gleichzeitig angezeigt werden sollen. Doch eine solche Anforderung gibt es nicht, daher genügt auch hier ein einziger Store.

5.6. Server-Logik (Controller) und URL-Schema

Da es sich bei STANLY_ACOS um eine Webapplikation handelt, wird die Kommunikation zwischen Client und Server vollständig über HTTP abgewickelt. Sämtliche server-

[8] Alle Elemente sind Instanzen der gleichen Model-Klasse.

[9] Beispiel: Anzeige der Datensätze 26–50 von 86, entsprechend aktuellem Filter und Sortierreihenfolge.

[10] Hierbei kommen weitere Klassen des ExtJS-Frameworks zum Einsatz, etwa ein *Proxy* für Erzeugung der Aufrufe zum Server sowie *Reader* und *Writer* zum Serialisieren und Deserialisieren der Model-Instanzen über die oben genannte generische Schnittstelle. All diese zusätzlichen Komponenten werden jedoch einheitlich auf Basis der Store-Konfiguration mit konfiguriert.

[11] Diese Eigenheit macht für die praktische Verwendung kaum einen Unterschied, spielt aber mit anderen Mechanismen des Frameworks, etwa dem Class-Loader, besser zusammen.

seitigen Komponenten sind von einer einzigen *Basis-URL* aus erreichbar.[12] Diese hat in der Regel folgende Form:

http*(s)*://*Name der VM*/stanly_acos_bnl/

Im Folgenden werden alle URLs relativ zur Basis-URL angegeben.

Die Basis-URL wird ausschließlich durch die Konfiguration der Systemumgebung vorgegeben (siehe Anhang C), die Applikation macht hierüber keine Annahmen und ist entsprechend flexibel. Die Struktur unterhalb der Basis-URL wird jedoch fest durch die Applikation vorgegeben und ist nicht konfigurierbar. Dieser feste URL-Namensraum ist zunächst einmal entsprechend der in Abschnitt 3.2, Abbildung 7 aufgeführten Komponenten unterteilt, die vom Web-Server ausgeliefert werden:[13]

URL	Komponente
app/	Applikation
xmpp/	XMPP-Server
ows/	Karten-Server
mapproxy/	Karten-Cache

Für den Prototyp wurde nur die Applikation erweitert. Diese basiert auf dem Zend-Framework, welches das Architekturmuster Model-View-Presenter umsetzt, auch wenn es sich selbst als Model-View-Controller bezeichnet (wie bereits in Abschnitt 3.2 diskutiert). STANLY_ACOS folgt der Konvention des Zend-Frameworks, den URL-Namensraum entsprechend der serverseitigen Controller aufzuteilen. Für den BNL-Prototyp wurden die vorhandenen Controller nicht erweitert, sondern ein neuer Controller mit entsprechendem URL-Namensraum hinzugefügt:

URL	Serverseitiger Controller
app/main/	MainController
app/sysadmin/	SysadminController
…	…
app/bnl/	BnlController

[12] Dies spielt sehr gut mit der Same-Origin-Policy des Browsers zusammen. Zudem vereinfacht es die Administration, da zum Beispiel in der Firewall oder im SSL-Gateway (siehe Anforderung 41) für die gesamte Applikation nur eine einzige URL freigegeben und verwaltet werden muss.

[13] Dies ist eine starke Vereinfachung. Auf oberster Ebene gibt es viele weitere URL-Bereiche. Ein Beispiel sind nach außen verfügbare Dienste wie *kpi/*, die nicht direkt vom Client, sondern von anderen Applikationen der DFS genutzt werden. Ein anderes Beispiel sind die zum Debugging benötigten Einzeldateien der clientseitigen Libraries. Weiterhin wurde der GML-Server in der Tabelle ausgelassen: Dieser stellt mehrere Services bereit, über unterschiedliche URL-Bereiche, die für den Prototyp aber allesamt nicht relevant sind.

Dieser befindet sich in folgender Datei:

- server/app/
  - Controller/
    - BnlController.php

Der Controller hat allgemein die Aufgabe,

- auf die eingehenden HTTP-Anfragen zu reagieren,
- die Überprüfung der nötigen Zugriffsrechte sicherzustellen (mehr dazu in Abschnitt 5.15),
- die mitgesendeten GET- und POST-Parameter zu prüfen und zu interpretieren,
- entsprechende Aktionen in den serverseitigen Models anzustoßen, und
- das Resultat an den richtigen View weiterzureichen, der daraus eine HTML-Seite, Excel-Tabelle oder ähnliches generiert.

Der BnlController kürzt viele dieser Schritte ab, wie fast alle Controller in STANLY_ACOS. Die klassischen CRUD-Aktionen werden 1:1 auf entsprechende Datenbankoperationen abbildet, wobei ein generisches CRUD-Model (Abschnitt 5.14) anstelle spezialisierter Models zum Einsatz kommt. Für die Darstellung des Resultats wird gar kein klassischer View bemüht, sondern die Daten werden direkt im JSON-Format ausgegeben.

Darüber hinaus erfüllt der Controller zwei zusätzliche Aufgaben, die er ebenfalls vollständig an die Datenbank delegiert: Dies ist zum einen die Vergabe der BNL-Nummern während der Create-Aktion, was in SQL sehr klar und direkt ausdrückbar ist. Zum anderen ist es die Interpolation von geographischen Kreisen (Abschnitt 5.9), was über eine benutzerdefinierte Datenbankfunktion geschieht, da PostgreSQL dank der PostGIS-Erweiterung deutlich besser zugängliche GIS-Funktionen bereit stellt als die entsprechenden Libraries für PHP.

Es folgt eine Übersicht aller BnlController-Aktionen, über welche URLs sie kodiert werden, ob sie POST-Parameter entgegen nehmen und von welcher Client-Komponente[14] (und welcher Operation in der Komponente) sie ausgelöst werden:

URL	POST	Client-Komponente (Op)
app/bnl/create?table=bnl	✓	store.Bnl (Create)
app/bnl/update?table=bnl	✓	store.Bnl (Update)
app/bnl/delete?table=bnl	✓	store.Bnl (Delete)
app/bnl/read?table=bnl&*…Read-Parameter…*	—	store.Bnl (Read)
app/bnl/read?table=bnlcenter	—	store.Bnlcenter (Read)
app/bnl/read?table=bnltype	—	store.Bnltype (Read)
app/bnl/read?table=uom_horizontal	—	store.UomHorizontal (Read)
app/bnl/read?table=uom_vertical	—	store.UomVertical (Read)
app/bnl/interpolatecircle?center=…&radius=…	—	controller.Bnl (createCircle)

[14] Die *store.**-Komponenten werden in Abschnitt 5.5 erklärt, die *controller.Bnl*-Komponente in Abschnitt 5.10.

Das URL-Schema verdeutlicht nochmals, dass die Read-Aktion sehr generisch ist und die konkrete Tabelle einfach als GET-Parameter *table=…* entgegen nimmt. Die erlaubten Tabellennamen werden dabei aus Sicherheitsgründen durch den BnlController fest vorgegeben. Die Create-, Update- und Delete-Aktionen sind ebenfalls generisch, auch wenn sie im Prototyp nur in Bezug auf die Tabelle *bnl* zum Einsatz kommen.[15] Größere Datenmengen werden als POST-Parameter entgegen genommen, die genau wie das Resultat allesamt JSON-kodiert sind.[16] Das Zend-Framework wie auch das clientseitige ExtJS-Framework würden auch XML-kodierte Daten unterstützen, aber in beiden Frameworks wird JSON wegen seiner geringeren Komplexität bevorzugt.[17]

Eine wichtige Einschränkung des Prototypen ist, dass noch keine *Read-Parameter* unterstützt werden. Der wichtigste Read-Parameter wäre *filter=…*, der in einer JSON-Struktur einen beliebigen Filter kodiert. So könnte man unter anderem einen Zeitbereich angeben, um nur die Vorhaben eines bestimmten Tages einzuladen. Andere Read-Parameter wären die Paging-Parameter *start=…* und *limit=…*. Diese sind natürlich nur dann sinnvoll, wenn die Sortierreihenfolge fest ist. Dies ist bereits im Prototypen vorbereitet, indem dort stets eine bestimmte Sortierreihenfolge vorgegeben ist. Es wäre aber auch denkbar, dass der Client einen entsprechenden Sortier-Parameter *sort=…* übergibt, der in einer JSON-Struktur die gewünschte Sortier-Reihenfolge beschreibt. All diese Read-Parameter konnten aus Zeitgründen serverseitig nicht implementiert werden, und wurden für erste Präsentationen auch nicht benötigt, aufgrund der geringen Anzahl von Test-Datensätzen. Dennoch sind die entsprechenden Schnittstellen bereits vorhanden und vorbereitet.

5.7. Automatische Aktualisierung

STANLY_ACOS besitzt seit je her die Fähigkeit, Änderungen auf Serverseite mit einer sehr geringen Verzögerungszeit allen Clients mitzuteilen. Wird eine Luftraumbuchung erstellt, geändert, genehmigt oder abgelehnt, sehen dies alle beteiligten Akteure sofort. Da die Clients im Browser laufen, stellt dies eine besondere Herausforderung dar.[18] Anfangs in STANLY_MVPA (Abschnitt 1.4) geschah dies noch durch regelmäßiges Polling aller Clients an den Server. Später wurde der XMPP-Server *ejabberd* eingeführt, an den die Clients via BOSH angebunden sind (siehe Abschnitt 3.2). Diese Fähigkeit zur auto-

[15] Die übrigen Tabellen enthalten nur Stammdaten, wie in Abschnitt 5.4 erklärt.

[16] Für POST-Daten wurde bewusst auf die klassischen Kodierungen *application/x-www-form-urlencoded* und *multipart/form-data* verzichtet, da diese in erster Linie für die Übertragung der Inhalte von Web-Formularen entwickelt wurden, und keine direkte Unterstützung für komplexere, verschachtelte Datenstrukturen bieten. Die JSON-Kodierung *application/json* löst das Problem.

[17] XML kommt in STANLY_ACOS nur in externen Schnittstellen zum Einsatz, wie zum Beispiel *kpi/*, da hier die Vorteile von XML überwiegen, insbesondere die Existenz von standardisierten, weit verbreiteten Schema-Sprachen wie DTD und XML Schema.

[18] Das wird sich in Zukunft ändern, da sich mit *WebSocket* ein neuer W3C-Standard etabliert, der eine echte bidirektionale Kommunikation zwischen Browser und Server ermöglicht. [Wor12] Der Internet Explorer unterstützt dies seit Version 10 [Ray12], aber STANLY_ACOS muss bislang auch Version 9 unterstützen (Anforderung 43).

matischen Aktualisierung wird auch für den BNL-Prototyp benötigt.[19]

Es gibt in STANLY_ACOS auf Client- wie auf Serverseite eine zentrale Messaging-Schnittstelle zur Übermittlung beliebiger JSON-Nachrichten. Es liegt jedoch in der Verantwortung jedes einzelnen Moduls, sinnvolle Nachrichten zu versenden und entsprechend darauf zu reagieren. Diese Aufgaben werden von den client- und serverseitigen Controllern wahrgenommen:

- client/App/
  - module/bnl/
    - controller/
      - Bnl.js
- server/app/
  - Controller/
    - BnlController.php

Die Messaging-Schnittstelle abstrahiert von der konkreten Realisierung über XMPP und verzichtet bewusst auf viele Features, die das Protokoll eigentlich ermöglicht. Zum einen findet ein Nachrichtversand nur vom Server an die Clients statt, die anderen Nachrichtenwege werden nicht angeboten:

Nachrichtenweg	Verwendung	Begründung
Client → Client	—	nicht benötigt
Client → Server	—	bereits via HTTP möglich
Server → Client	✓	automatische Aktualisierung
Server → Server	—	nur ein Server (Anforderung 42)

Zum anderen gibt es keine Empfängerbeschränkung. Jede Nachricht erreicht alle Clients. Im Gegenzug sind die Nachrichten sehr kurz und enthalten keine sensiblen Informationen. Der Server informiert die Clients nur grob darüber, *dass* und *wo* sich etwas geändert hat. Die eigentlichen Informationen laden die Clients dann über normale HTTP-Aufrufe nach.

Konkret im Prototyp gibt es nur einen einzigen Nachrichtentyp *bnlReload*, der keine weiteren Parameter besitzt. Diese Nachricht wird nach jeder erfolgreichen Create-, Update- oder Delete-Aktion (siehe Abschnitt 5.6) ausgelöst. Der Vorteil ist, dass das Aktualisierungs-Protokoll dadurch sehr einfach[20] ist, und die Synchronität der Daten unter allen Umständen sicherstellt. Aber es ist auch sehr verschwenderisch, was bei genauerer Betrachtung des Informationsflusses klar wird:

1. Der Client sendet eine Änderung zum Server, etwa die Löschung eines Vorhabens.

[19] Bemerkenswert ist, dass dieses Feature in STANLY_ACOS so selbstverständlich geworden ist, dass es weder in der ursprünglichen Anforderungserhebung [Har12] bedacht wurde, noch in der eigenen. Erst viel später, während der Implementierung des Prototypen, fiel diese Lücke in den Anforderungen auf und wurde durch Anforderung 40 nachträglich geschlossen.

[20] Trotz seiner Einfachheit führt selbst dieses Aktualisierungs-Protokoll zu einigen Herausforderungen in der Benutzer-Interaktion. Mehr dazu in Abschnitt 5.10.

2. Der Server führt diese Änderung durch.
3. Der Server sendet die Nachricht *bnlReload* an alle Clients.
4. Alle Clients laden die Vorhabensliste und die Grunddaten neu vom Server.

Dies führt insbesondere bei der Client-Instanz zu einem Reload, die die Änderung selbst ausgelöst hat, obwohl sie keine Information dadurch gewinnt. Für den Benutzer bedeutet das nach jeder eigenen Änderung eine kleine, aber deutlich spürbare Verzögerung.

Die Lösung wäre ein ausgefeiltes Aktualisierungs-Protokoll, wie es an anderen Stellen von STANLY_ACOS zum Einsatz kommt. Das würde jedoch den Rahmen dieser Arbeit sprengen, da hierfür in STANLY_ACOS keine generischen Komponenten zur Verfügung stehen, sondern jedes Modul ein eigenes, spezialisiertes Aktualisierungs-Protokoll umsetzen muss.

5.8. Client-Views

Nach dem Einloggen in STANLY_ACOS und Betätigen des Menüeintrages „BNL“ öffnet sich rechts neben dem Menü ein neues Tab-Element, das BNL-Modul, wie in Abbildung 18 dargestellt. Dieses Tab-Element wird wie in MVC-Frameworks üblich durch einen View beschrieben. Dabei funktionieren die Views in ExtJS genauso wie in den klassischen nativen GUI-Bibliotheken.[21] Das heißt, der View beschreibt die Zusammensetzung der GUI aus den Komponenten des Frameworks, die überwiegend Container sind, welche wiederum kleinere Komponenten enthalten. Zur besseren Handhabung sind große Teile des BNL-Views in mehrere Unter-Views ausgelagert. Diese können vom übergeordneten View direkt eingebunden werden, als wären es Standard-Komponenten des Frameworks.

Die Zusammenstellung der Oberfläche aus kleineren Elementen erfolgt nicht über das direkte Spezifizieren einer HTML-Struktur, sondern alles geschieht auf Ebene der Komponenten. Das heißt, die Views in ExtJS sehen ausschließlich Framework-Komponenten und keine HTML-Elemente. Dies ist ein großer Unterschied von ExtJS zu vielen anderen Web-Frameworks. Es ist die große Stärke von ExtJS, und zugleich dessen größte Schwäche. Denn auf unterster Ebene bilden ExtJS-Komponenten natürlich wieder ein HTML-Dokument. Das heißt, sie bestehen aus ganz normalen HTML-Elementen, auch wenn diese via CSS besonders gestylt und damit kaum wiederzuerkennen sind. Jede Verschachtelungsebene im View entspricht einem Vielfachen dieser Verschachtelungstiefe im tatsächlichen HTML-Dokument. Die daraus resultierende Komplexität in der Dokumentenstruktur führt nicht nur zu Performance-Problemen bei älteren Browsern,[22] sondern erschwert auch das Debugging bei Darstellungsproblemen erheblich. Aus besonders vielen Elementen besteht dabei eine dynamische Tabelle (*Grid*-Komponente), die konkret im BNL-Modul für die Vorhabensliste zum Einsatz kommt.

[21] Gemeint sind GUI-Bibliotheken wie Qt, GTK+ oder Swing.

[22] Auch bei aktuellen Browsern führt dies zu Performance-Problemen, wenn die GUI sehr viele Elemente hat, oder wenn tausende von Zeilen in ein Grid geladen werden.

Die Views des BNL-Moduls sind in folgenden Dateien definiert:

- client/App/
  - module/bnl/
    - view/
      - Bnl.js
      - BnlForm.js
      - BnlGrid.js
      - BnlContextMenu.js

Dem MVC-Muster entsprechend beschreiben sie lediglich den Aufbau der Oberfläche und enthalten keine Logik zur Benutzerführung, denn dafür ist der Controller zuständig (siehe Abschnitt 5.10). Entsprechend sind die Views rein deklarativ formuliert und könnten fast durch eine JSON-Datenstruktur ersetzt werden.

Nur in absoluten Ausnahmefällen, wenn die deklarativen Möglichkeiten versagen, enthalten Views Programm-Code. Konkret im BNL-Modul ist dies an einer Stelle notwendig: Für die Tabellenspalte *Center* wird eine benutzerdefinierte Rendering-Funktionen verwendet, die das zum *center* zugehörige *center_label* (siehe Abschnitt 5.4) mit Hilfe von *store.Bnlcenter* auflöst, sodass dort beispielsweise *GG* statt *EDGG* angezeigt wird. Idealerweise würde das Framework für diese oft benötigte Standardaufgabe einen generischen, rein deklarativ ansteuerbaren Mechanismus zur Verfügung stellen. Doch das gibt es derzeit nicht, daher kommt eine benutzerdefinierte Rendering-Funktionen zum Einsatz.

Die View-Definitionen erhalten nicht nur offizielle ExtJS-Attribute wie Breite, Höhe, Aktiviert/Deaktiviert und Beschriftungen. Die Komponenten können zusätzlich mit selbstdefinierten Attributen annotiert werden, um diese im Controller auszuwerten. Konkret im BNL-Modul kommen die selbstdefinierten Attribute *disabledInModes* und *hiddenInModes* zum Einsatz. Mehr dazu in Abschnitt 5.10.

Abbildung 18 zeigt anhand eines Screenshots, welche Teile der GUI von welchem View beschrieben werden. Die Views werden im Folgenden erläutert.

App.module.bnl.view.Bnl ist der Haupt-View, der das gesamte Tab-Element repräsentiert. Er beschreibt die obere Button-Leiste und enthält alle übrigen Views als direkte Kind-Elemente. Zudem spezifiziert er, wie die Unter-Views angeordnet sind: links das *BnlForm* mit fester Breite, in der Mitte das *BnlGrid* mit automatischer Breite und rechts das *MapPanel* mit fester Breite, die jedoch vom Benutzer über eine *Splitter*-Komponente veränderbar ist.

App.module.bnl.view.BnlForm repräsentiert das Formular zur Eingabe und Bearbeitung von BNL-Vorhaben. Es beschreibt genau die Felder aus Anforderung 1 mit den in Abschnitt 5.4 erläuterten Abweichungen. Einige Felder werden durch Auswahllisten (*ComboBox*-Komponenten) realisiert, deren zugehöriger Store jeweils direkt im View festgelegt wird. Zum Beispiel wird der Vorhabens-Typ über eine *ComboBox* ausgewählt, die auf den Store *store.Bnltype* konfiguriert ist. Dieser enthält die Liste aller gültigen Vorhabens-Typen, die entsprechend in der Auswahlliste angeboten werden.

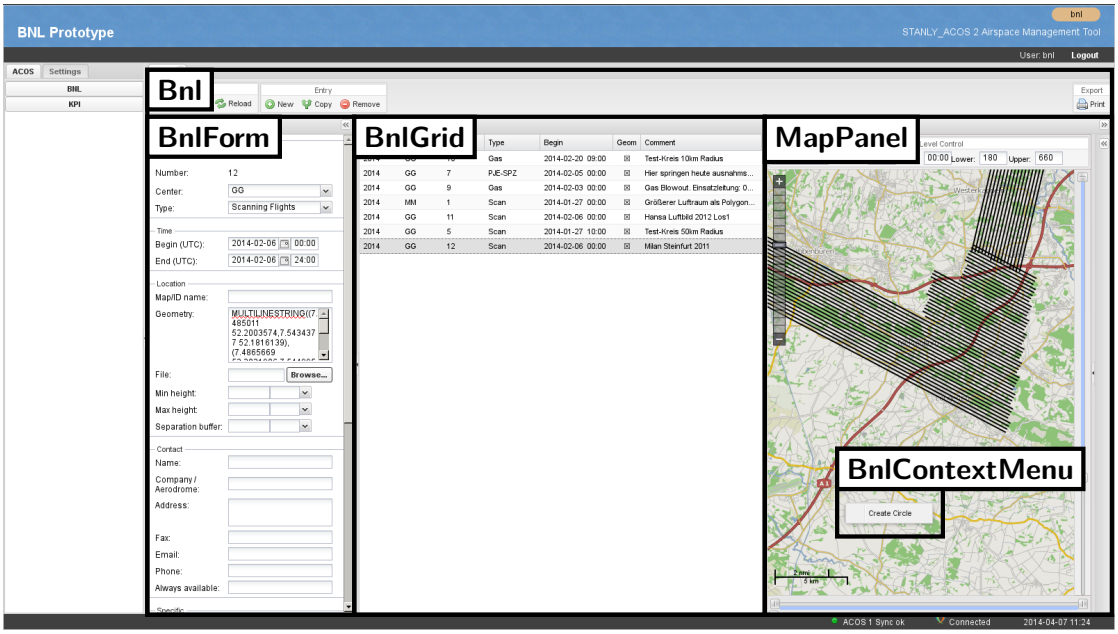


Abbildung 18.: Screenshot des Prototypen, der die Struktur der clientseitigen Views veranschaulicht. Der Kürze halber werden nur die Klassennamen der Views ohne Namespaces gezeigt, zum Beispiel *BnlForm* statt *App.module.bnl.view.BnlForm*.

App.module.bnl.view.BnlGrid repräsentiert die Vorhabensliste. Dieser View beschreibt nur eine einzige Komponente, eine *Grid*-Komponente, die jedoch sehr viele Konfigurationsparameter enthält. Zum einen wird das *Grid* auf den Store *store.Bnl* konfiguiert, der alle aktuell im Client geladenen BNL-Vorhaben enthält. Zum anderen wird die Darstellung für sämtliche Tabellen-Spalten festgelegt, sichtbare wie unsichtbare. Da die Vorhabensliste schnell unübersichtlich wäre, wenn alle Spalten gleichzeitig angezeigt würden, ist zunächst nur eine bestimmte Menge von Spalten sichtbar. Die unsichtbarn Spalten können zur Laufzeit vom Benutzer eingeblendet werden. Störende Spalten kann der Benutzer ausblenden. Eine clientseitige Umsortierung nach einer bestimmten Spalte kann ebenfalls über die *Grid*-Komponente ausgelöst werden, wobei die eigentliche Umsortierung durch den Store erfolgt.

App.widget.map.MapPanel ist, wie der Namespace *App.widget.map* bereits verrät, kein View des BNL-Moduls. Da diese Komponente jedoch einen großen Bereich der Benutzeroberfläche einnimmt, soll sie an dieser Stelle dennoch explizit aufgeführt werden. Es handelt es sich um eine bereits in STANLY_ACOS vorhandene Komponente zur Darstellung von Karten (GIS-Client), die vom BNL-Modul direkt und ohne weitere Modifikation genutzt wird.

Sie basiert auf der OpenLayers-Bibliothek [Ope12] und kann verschiedene Vektor-Kartenelemente (*Features*) in mehreren Schichten (*Layers*) darstellen. Dabei kann

sie die *Features* flexibel stylen und zudem mit ihnen interagieren. Als Hintergrund kann sie wahlweise eine schlichte helle Karte, eine schlichte dunkle Karte oder eine OpenStreetMap-Karte anzeigen.[23] Die Darstellung erfolgt in der in STANLY_ACOS üblichen Projektion EPSG:3857 (Pseudo-Mercator [OGP14b], auch Google-Projektion genannt), doch es werden auch viele andere Projektionen unterstützt.

Vom BNL-Modul nicht verwendet, aber dennoch prinzipiell vorhanden, ist die Möglichkeit, ein *Feature* anzuwählen und den zugehörigen Eintrag in der Vorhabens-Liste automatisch auszuwählen. Ebenfalls keine Verwendung finden die vorgesehenen Höhen- und Zeitfilter, die über Textfelder wie auch Schieber-Elemente (*Slider*) eingestellt werden können.[24]

App.module.bnl.view.BnlContextMenu repräsentiert das Kontextmenü, das sich bei einem Rechtsklick auf die Karte öffnet. Dieses enthält nur einen einzigen Menüeintrag *Create Circle*, dessen Funktionalität in Abschnitt 5.9 erklärt wird. Wie die anderen Views ist auch dieser ein Unter-View von *App.module.bnl.view.Bnl*, obwohl er als Kontextmenü eigentlich keine Elternkomponente (*Parent Node*) benötigt. Er kann überall erscheinen, innerhalb wie außerhalb seiner *Parent Node*. Jedoch sorgt die vorhandene *Parent Node* dafür, dass sein Lebenszyklus automatisch verwaltet wird. Sonst müsste er im Controller manuell erzeugt und zerstört werden, unter Beachtung aller denkbaren Randfälle.

Die Klassennamen der Views tragen alle den Präfix *Bnl…*, obwohl sie bereits in einem eigenen Namespace *App.module.bnl.view* liegen. Dies ist leider notwendig, weil die View-Klassen in ExtJS nicht über ihren voll qualifizierten Klassennamen, sondern über einen Kurznamen identifiziert werden, den sogenannten *xtype*.[25] In STANLY_ACOS gilt die Konvention, dass der *xtype* einfach dem Klassennamen entspricht,[26] daher müssen die Klassennamen aller Views über sämtliche Module hinweg eindeutig sein. Dies wird am einfachsten durch einen Präfix sichergestellt, der dem Modulnamen entspricht. Der Präfix *Bnl…* übernimmt letztendlich also die Rolle eines *xtype*-Namespaces.

[23] Mit entsprechender Konfiguration kann sie auch beliebiges anderes Kartenmaterial einblenden, sofern es fertig gerendert über einen WMS (Web Map Service), TMS (Tile Map Service) oder WMTS (Web Map Tile Service) ausgeliefert wird. Hiervon macht der Prototyp jedoch keinen Gebrauch.

[24] Diese wären für zukünftige Entwicklungen sicher interessant und nützlich. Allerdings findet die entsprechende Kopplung zwischen den Filtern und dem Store (hier: *store.Bnl*) nicht automatisch statt, sondern muss derzeit in STANLY_ACOS spezifisch für jedes Modul neu implementiert werden.

[25] Der *xtype*-Mechanismus von ExtJS ermöglicht es, auf komplexe, verschachtelte Konstruktor-Aufrufe zu verzichten. So wird das *BnlForm* nicht direkt über `new App.module.bnl.view.BnlForm({…})` instanziiert, sondern es wird nur ein Konfigurations-Objekt `{xtype: "BnlForm", …}` spezifiziert. Das ExtJS-Framework löst dann den *xtype* auf und ersetzt das Konfigurations-Objekt durch eine entsprechende *BnlForm*-Instanz.

[26] Dies ist bereits eine Verbesserung zur Situation der ExtJS-Standardkomponenten, deren *xtype*s gar keiner einheitlichen Konvention folgen. Beispiel: *Ext.form.field.Text* → *textfield*, aber *Ext.form.field.ComboBox* → *combobox* (nicht *comboboxfield*).

5.9. Interpolation von geographischen Kreisen

Laut Anforderung 10 sollen in den Prototypen geographische Kreise[27] eingegeben und auf der Karte dargestellt werden können. Das ist jedoch in OpenLayers (siehe Abschnitt 5.8) nicht direkt möglich, was auch für die meisten anderen GIS-Clients gilt. Zwar können die GIS-Clients um einen gegebenen Punkt herum einen Kreis zeichen, doch dieser erscheint dann auf jeder Karte als exakter Kreis, statt der Projektion entsprechend verzerrt zu werden. Je nach Radius und Projektion kann diese Ungenauigkeit tolerierbar sein oder auch nicht.

Die Standardlösung für dieses Problem besteht darin, komplexe Objekte serverseitig zu interpolieren, das heißt, durch Polygone anzunähern. Diese Lösung kommt auch im Prototyp zum Einsatz. Hierbei wird jeder geographische Kreis durch ein regelmäßiges 64-Eck angenähert. Dieses kann von sämtlichen GIS-Clients problemlos dargestellt werden, so auch OpenLayers, und ist für alle praktischen Belange mehr als genau genug.

Die eigentliche Berechnung findet in der benutzerdefinierten Datenbank-Funktion

interpolate_circle(*center*, *radius*)

statt, die das Zentrum des Kreises in *EPSG:4326*-Koordinaten (Längen- und Breitengrad, siehe Anforderung 7) sowie den Radius in Metern entgegen nimmt. Das Ergebnis liefert sie als Polygon in *EPSG:4326*-Koordinaten zurück. Sie ist in folgender Datei definiert:

- server/db/
  - bnl/
    - func/
      - interpolate_circle.sql

Datenbankseitig stehen dank *PostGIS* [Ope14] fertige Interpolations-Funktionen zur Verfügung. Doch leider arbeiten diese innerhalb eines lokalen, zweidimensionalen Koordinatensystems, ohne die Verzerrungen der Projektion auszugleichen. Dies gilt nicht nur für PostGIS, sondern für nahezu alle gängigen GIS-Systeme. Daher musste für den Prototypen die Entscheidung getroffen werden, welche der folgenden Optionen das kleinere Übel darstellt:

1. eine eigene, exakte Implementierung der Interpolation
2. eine Kombination aus Standardfunktionen – jedoch innerhalb einer Hilfs-Projektion, in der Kreise möglichst wenig verzerrt werden

[27] Genauer gesagt geht es um geographische Kreis-Scheiben, da nicht nur der Rand, sondern auch das Innere dazu zählt. Eine geographische Kreis-Scheibe um dem Mittelpunkt M mit Radius r ist definiert als die Menge aller Punkte P, deren Abstand von M höchstens r beträgt. Als Abstand gilt dabei nicht der gewöhnliche (euklidische) Abstand zwischen M und P im dreidimensionalen Raum, sondern der Abstand auf der Erdoberfläche, das heißt, die Länge der kürzesten Verbindungskurve von M nach P auf dem in *WGS 84* definierten Referenzellipsoiden.

Im Prototyp ist die zweite Option implementiert, da diese mit einem deutlichen geringerem Aufwand verbunden ist, ohne dass die Genauigkeit zu sehr leidet. Als Hilfs-Projektion kommt EPSG:25832 [OGP14a] zum Einsatz, die für diesen Zweck sehr gut geeignet ist.[28] Es folgt der genaue Ablauf des Verfahrens:

	GIS-Funktion	Beschreibung
1.	ST_SetSRID	Markierung des gegebenen Mittelpunktes als EPSG:4326, falls noch nicht geschehen
2.	ST_Transform	Koordination-Transformation des Mittelpunktes von EPSG:4326 nach EPSG:25832
3.	ST_Buffer[29]	Erzeugung eines Pufferbereiches (Polygon, 64 Ecken) um den Mittelpunkt herum mit dem gegebenen Radius
4.	ST_Transform	Koordination-Transformation des Polygons von EPSG:25832 nach EPSG:4326

Die Datenbank-Funktion ist vollständig in SQL implementiert und verwendet ausschließlich standardisierte GIS-Funktionen[30] sodass sie möglichst wenig PostGIS-spezifisch ist.

Dabei ist die Anzahl der Punkte des Polygons (64) bewusst *kein* Parameter von *interpolate_circle*, sondern innerhalb der Funktion hart codiert. Es gehört zur Aufgabe von *interpolate_circle*, eine mehr als ausreichende Genauigkeit sicherzustellen. Sollte dieser Wert variieren müssen, wäre es immer noch Aufgabe von *interpolate_circle*, abhängig vom Radius eine geeignete Anzahl von Interpolationspunkten festzulegen.

5.10. GUI-Steuerung und Benutzerführung

Das Kernstück des clientseitigen BNL-Moduls ist der Controller, der die gesamte GUI-Steuerung vornimmt. Hierbei ist das Design-Ziel, den gesamten Kontrollfluss der Benutzerführung an zentraler Stelle nachvollziehbar zu machen. Der Controller hat naturgemäß die höchste Komplexität und muss durch Hilfsklassen wie Models, Stores (Abschnitt 5.5) und Views (Abschnitt 5.8) entlastet werden. Diese haben fast keine Interaktionen untereinander[31], sondern stets Schnittstellen zum Controller hin: durch Methoden, die der Controller aufrufen kann, und Events, auf die der Controller reagieren kann. Dies ist klassisches MVC-Design: Die Komplexität der GUI-Steuerung wird nicht aufgeteilt, sondern isoliert. Der Controller *App.module.bnl.controller.Bnl* befindet

[28] EPSG:25832 basiert auf ETRS89 (Europäisches Terrestrisches Referenzsystem 1989), welches wiederum auf der UTM-Abbildung basiert. [Lan12, S. 14]

[29] Diese GIS-Funktion kann nicht nur einen Puffer um einen einzelnen Punkt erzeugen, sondern um eine beliebige Geometrie herum. Somit könnte sie in einer zukünftigen Entwicklung auch zur Berechnung und Darstellung des in Anforderung 45 beschriebenen Staffelungsbereiches genutzt werden.

[30] Gemeint ist der OGC-Standard *SFA-SQL* [Ope10a], den PostGIS implementiert. [Ope14, S. 29] Dabei wird konsequent der Präfix *ST_* verwendet, um Kompatibilität zum ISO-Standard *SQL/MM-Spatial* zu wahren. [Ope10a, S. 70]

[31] Die Ausnahme sind triviale, leicht nachvollziehbare Interaktionen, wie zum Beispiel die Aktualisierung eines *Grid* (zum Beispiel der Vorhabens-Liste), wenn sich der zugehörige *Store* ändert.

sich in folgender Datei:

- client/App/
  - module/bnl/
    - controller/
      - Bnl.js

Während der Entwicklung wurden einige generische Methoden des Controllers in eine Hilfsklasse ausgelagert, mehr dazu in Abschnitt 5.11.

Konkret im BNL-Modul hat der Controller die Aufgabe, den Datenfluss zwischen Formular, Liste, Karte, Models und Stores zu koordinieren: Betätigt der Benutzer den Button *New*, muss das Formular entleert und mit sinnvollen Default-Werten belegt werden. Dann muss das erste Formularfeld fokussiert werden.[32] Während der Bearbeitung muss die aktuelle Geometrie sofort auf der Karte dargestellt werden, auch wenn das aktuelle Vorhaben noch nicht gespeichert wurde. Betätigt der Benutzer den Button *Add* (oder bei einer späteren Bearbeitung den Button *Save*), so muss der Inhalt des Formulars in ein entsprechendes Model übertragen werden, das gegebenenfalls in den Store eingefügt wird, der daraufhin die entsprechende Aktionen im Server auslösen muss. Wird ein anderes Vorhaben in der Liste ausgewählt, müssen neue Daten vom entsprechenden Model in das Formular geladen werden. Zudem muss die enthaltene Geometrie in der Karte hervorgehoben werden. Wird ein Vorhaben bearbeitet und der Benutzer betätigt nicht *Save*, sondern *Cancel*, so muss der alte Inhalt des Models in das Formular übertragen werden. Und so weiter.

Für jeden einzelnen dieser Schritte bietet das ExtJS-Framework fertige Methoden[33] an, doch für die gesamte Orchestrierung dieser Abläufe gibt es keinen fertigen Mechanismus. Möglicherweise wird es so etwas auch niemals geben, da bereits eine kleine Änderung in den Anforderungen dazu führen kann, dass sich viele Details der Benutzerführung ändern.

Konkret in den Anforderungen des Prototypen gibt es nur wenige Vorgaben zur Benutzerführung und entsprechend viele Freiheiten. Daher fiel die Entscheidung, die Benutzerführung im BNL-Modul so zu gestalten, dass sie in der Programmierung möglichst wenig Probleme bereitet. Das heißt, in jedem Zustand werden so viele GUI-Elemente wie möglich gesperrt, sodass es möglichst wenige Sonderfälle und damit möglichst wenige potentielle Fehlerquellen gibt. Beispiele hierfür sind:

- Die Auswahlliste wird sofort gesperrt, wenn am aktuellen Vorhaben über das Formular eine Änderung durchgeführt wurde (*dirty*). So braucht sich der Controller nicht um den Sonderfall zu kümmern, dass der Benutzer ein anderes Vorhaben

[32] Hierbei kommt die in Abschnitt 5.11 beschriebene Methode *getFirstEditableField()* zum Einsatz.

[33] Zum Beispiel gibt es fertige Methoden zum Befüllen aller Formularfelder mit den Daten eines Models, oder umgekehrt dem Setzen aller Model-Daten aus einem Formular. Dies funktioniert, da die in Abschnitt 5.5 beschriebenen generischen Schnittstellen nicht nur für Models und Stores, sondern auch für Formularfelder existieren. Im Detail haben diese Mechanismen dennoch ihre Tücken, mehr dazu in Anhang D.

anwählt, während das Formular *dirty* ist. Der Benutzer muss entweder den Button *Save* oder den Button *Cancel* betätigen, bevor er sich einem anderen Vorhaben zuwenden kann.
- Die Buttons *Save* und *Cancel* stehen nur dann zur Verfügung, wenn der Benutzer ein vorhandenes Vorhaben editiert hat (Formular ist *dirty*). Macht er diese Änderung von Hand rückgängig (indem er in die bearbeiteten Felder wieder die alten Werte einträgt), so verschwinden die Buttons sofort wieder. Dadurch gibt es bei der Behandlung von *Save* weder den Sonderfall, dass es gar keine Änderung zu speichern gibt, noch den Sonderfall, dass gerade überhaupt kein Vorhaben ausgewählt wurde.
- Ist kein Vorhaben in der Liste ausgewählt, so wird das Formular gesperrt. Möchte der Benutzer ein neues Vorhaben anlegen, muss er zuvor explizit den *New*-Button betätigen.
- Die Vorhabensliste kann über den *Reload*-Button nur dann vom Server neu geladen und aktualisiert werden, wenn der Benutzer gerade kein Vorhaben bearbeitet (Formular nicht *dirty*) und nicht gerade dabei ist, ein neues Vorhaben anzulegen.

Solche Sperrungen sind zwar mit gewissen Einschränkungen für den Benutzer verbunden. Doch dem gegenüber steht ein großer Gewinn an Klarheit für beide Seiten: Der Arbeitszustand der Software ist jederzeit für den Benutzer klar erkennbar, und die Intention des Benutzers ist jederzeit für die Software zweifelsfrei erkennbar. Dies kommt insbesondere der in Abschnitt 5.7 beschriebenen automatischen Aktualisierung zugute.

Die Umsetzung dieser Sperrungen war zunächst ein Problem, denn abhängig von mehreren Parametern:

- ob in der Vorhabensliste ein Eintrag ausgewählt ist oder nicht
- ob das Formular *dirty* ist oder nicht
- …

müssen viele GUI-Elemente modifiziert werden, und zwar potentiell auf zwei verschiedene Arten und Weisen:

- Sie müssen gesperrt (*disabled*) oder wieder entsperrt werden.
- Sie müssen unsichtbar (*hidden*) oder wieder sichtbar gemacht werden.

In der ersten, naiven Umsetzung führte dies zu einem komplizierten und äußerst fehleranfälligen Regelsystem. Glücklicherweise konnte die Komplexität durch zwei einfache Refactoring-Schritte deutlich reduziert werden.

Zuerst wurden die vielen theoretisch möglichen Parameterzustände auf nur 4 Zustände reduziert, die intuitiv gut erfassbar sind und im Folgenden als *Modi* bezeichnet werden:

Modus	Interpretation
unselected	kein Vorhaben ausgewählt
selectedClean	Vorhaben ausgewählt
selectedDirty	Vorhaben ausgewählt und im Formular abgeändert
create	neues Vorhaben im Formular angelegt

Diese Reduktion war möglich, da zum Beispiel die meisten Parameter irrelevant sind, solange gar kein Vorhaben in der Liste ausgewählt ist. Zudem ist es während des Anlegens eines neuen Vorhabens vollkommen unwichtig, ob das Formular *dirty* ist oder nicht.[34]

Im zweiten Schritt wurde jede vom Modus abhängige Komponente im View durch selbstdefinierte Attribute (siehe Abschnitt 5.8) annotiert:

Attribut	Inhalt
disabledInModes	Liste der Modi, in denen die Komponente deaktiviert sein muss
hiddenInModes	Liste der Modi, in denen die Komponente unsichtbar sein muss

Dabei wurden die meisten Komponenten ausschließlich mit *disabledInModes* annotiert, denn *hiddenInModes* führt zum plötzlichen Verschwinden und Auftauchen von GUI-Elementen und muss daher sehr sparsam verwendet werden.

Der Mechanismus zum Modifizieren der Komponenten anhand ihrer Annotationen ist sehr generisch und wurde daher in die Hilfsklasse *App.util.Container* ausgelagert.[35] Hochgradig applikationsspezifisch hingegen ist die Entscheidung, welche Modi es geben soll und wie diese definiert sind. Diese Funktionalität verbleibt daher im Controller.

Die automatische Aktualisierung (Abschnitt 5.7) wirft einige besondere Fragen in der der Benutzerführung auf. Was genau soll passieren, wenn der Client eine *bnlReload*-Nachricht erhält? Die Anforderungen lassen hier großen Spielraum, daher lag auch hier der Fokus darauf, eine Lösung zu finden, die in der Programmierung möglichst wenig Probleme bereitet: In den Modi *unselected*, *selectedClean* und *create* wird einfach die Vorhabens-Liste *store.Bnl* einfach neu geladen, wobei im Modus *create* dafür Sorge getragen werden muss, dass das Formular nicht neu geladen wird. Der Modus *selectedDirty* hingegen muss gesondert behandelt werden. Ein naives Neuladen von *store.Bnl* würde in ExtJS die Verbindung zwischen Formular und ausgewähltem Listen-Eintrag zerstören. Dieser Effekt könnte durch das Neuladen des Formulars verhindert werden, aber das würde die ungespeicherten Änderungen des Benutzers zunichte machen. Der einfachste Ausweg besteht darin, dem Benutzer vorerst nur eine Warnung anzuzeigen, dass sich etwas geändert hat.[36] Erst nach dem Betätigen von *Save* oder *Cancel* (wenn sich das BNL-Modul wieder im Modus *selectedClean* befindet) wird das Neuladen von *store.Bnl* ausgelöst. Dies wird im Controller durch ein einfaches Flag *pendingReload* koordiniert.

Wie in Abschnitt 5.1 erwähnt ist dieser Mechanismus natürlich kein Ersatz für ein ausgefeiltes Konzept zur Konfliktbehandlung bei paralleler Bearbeitung. Aber er erhöht die Sichtbarkeit potentieller Konflikte und entschärft zumindest die Gefahr des *unbemerkten* Überschreibens wichtiger Informationen.

Die Geometrien der Vorhaben werden stets in ein mehrzeiliges Textfeld eingegeben, dessen Inhalt zuerst geparst (Abschnitt 5.13) und dann wie in Anforderung 47 gefordert in Echtzeit auf der Karte dargestellt wird. Dies geschieht über einen extra Layer in der Karte, der besonders gestylt ist (dickere schwarze Linien zur Hervorhebung). Zugleich

[34] Aus diesem Grund gibt es keine weitere Unterteilung des *create*-Modus in *createClean* und *createDirty*.
[35] Es handelt sich um die in Abschnitt 5.11 beschriebene Methode *applyMode()*.
[36] Die exakte Warnung lautet: *Some information has changed on server side while you are editing. It will be reloaded as soon as you save or cancel.*

wird dabei das aktuelle Vorhaben aus dem eigentlichen BNL-Layer ausgeblendet, damit dessen Geometrie während der Bearbeitung nicht hinter der neuen Geometrie noch sichtbar ist und irritiert.

Darüber hinaus gibt es die Möglichkeit, eine Geometrie über das Kontextmenü der Karte festzulegen. Das Menü enthält derzeit nur einen einzigen Menüeintrag *Create Circle* zum Erzeugen eines Kreises um den angewählten Mittelpunkt herum. Dabei werden die Pixel-Koordinaten der Maus in geographische Koordinaten umgerechnet, die den Kreis-Mittelpunkt darstellen. Zusätzlich wird der Radius abgefragt und schließlich die Kreis-Geometrie interpoliert (siehe Abschnitt 5.9). Diese wird automatisch in das Geometrie-Textfeld eingetragen, welches dann für die Darstellung auf der Karte sorgt.

Dieses Verfahren hat den Vorteil, dass es sehr einfach ist und die gesamte Funktionalität über das Geometrie-Feld abwickelt. Der Nachteil ist jedoch, dass dabei die Rohdaten des Kreises (Mittelpunkts-Koordinaten und Radius) verloren gehen.

5.11. Generische Hilfsfunktionen zur GUI-Steuerung

Während der Entwicklung des clientseitigen Controllers (Abschnitt 5.10) wurden einige generische Methoden in eine neue Hilfsklasse *App.util.Container* ausgelagert. Da diese nicht spezifisch für BNL ist, befindet sie sich nicht im Namespace *App.module.bnl*, sondern in *App.util*:

- client/App/
  - util/
    - Container.js

Sie enthält unter anderem die Methode *getFirstEditableField()*, die das erste zu fokussierende Eingabefeld eines Formulars automatisch ermittelt.[37] Im BNL-Modul ändert sich dieses zur Laufzeit: Bei neu erstellten Vorhaben ist *Year* das erste Eingabefeld. Beim nachträglichen Bearbeiten hingegen darf das Jahr nicht mehr verändert werden, das Eingabefeld entfällt und stattdessen muss das Auswahlfeld für *Center* fokussiert werden. Diese Felder könnten natürlich auch hart codiert werden, aber die Funktion *getFirstEditableField()* löst das Problem generisch und macht den Controller damit unempfindlich gegenüber Umgestaltungen des Formulars.

Weiterhin enthält die Klasse die Methode *applyMode()*, welche einen View[38] sowie den aktuellen Modus (siehe Abschnitt 5.10) als Parameter erhält. Sie ermittelt automatisch alle mit *disabledInModes* oder *hiddenInModes* annotierten Komponenten des Views. Jede dieser Komponenten wird dann deaktiviert oder aktivert, jenachdem, ob sich der aktuelle Modus in deren *disabledInModes*-Liste befindet oder nicht. Analog werden die Komponenten unsichtbar oder wieder sichtbar gemacht, jenachdem, ob der aktuelle Modus in der jeweiligen *hiddenInModes*-Liste aufgeführt ist.

[37] Das ist in ExtJS nicht ganz trivial: Gesucht ist nicht einfach das erste Formularfeld, sondern das erste, das editierbar, nicht deaktiviert und kein reines Anzeigefeld (*Ext.form.field.Display*) ist.

[38] Konkret im BNL-Modul wird einfach der Hauptview *Bnl* übergeben, siehe Abschnitt 5.8.

5.12. Eingabe von Datum und Uhrzeit

Für die Eingabe von Datum und Uhrzeit existiert in STANLY_ACOS bereits eine Komponente *DateTimeField*, die jedoch hauptsächlich für Filtereinstellungen angedacht ist und für die Formular-Eingabe im Prototyp nicht direkt verwendet werden kann. Daher wurde für den Prototyp eine eigene GUI-Komponente für die kombinierte Eingabe von Datum und Uhrzeit erstellt, die zur Vermeidung von Namenskollisionen vorerst *DateTimeFieldBnl* genannt wurde und sich in folgender Datei befindet:

- client/App/
  - widget/
    - DateTimeFieldBnl.js

5.13. Parsen von Geometrien

Die Eingabe von Geometrien wird durch zwei Klassen realisiert, einer generischen Parser-Klasse und einer GUI-Komponente:

- client/App/
  - util/
    - GeometryParser.js
  - widget/
    - GeometryField.js

Der Parser erkennt automatisch die folgenden Formate:

- GeoJSON (über die OpenLayers-Bibliothek [Ope12])
- WKT (über die OpenLayers-Bibliothek [Ope12])
- OVL (selbst implementiert)
- in beliebigen Text eingebettete Koordinaten (selbst implementiert). Dies ist zum Kopieren von Daten aus Excel-Tabellen heraus gedacht.

Die GUI-Komponente stellt ein Textfeld bereit, dessen Inhalt bei jeder Änderung automatisch geparst wird. Parserfehler werden dem Benutzer im gleichen Stil wie andere Formular-Validierungsfehler angezeigt.

5.14. Generisches CRUD-Model

Der ORM des serverseitig verwendeten Zend-Frameworks [Zen14] verlangt für jede Tabelle die Definition einer separaten Model-Klasse. Das ist für viele Anwendungen durchaus sinnvoll, in denen diese Klassen um Geschäftsregeln und andere Funktionalitäten erweitert werden müssen.

Es passt jedoch weniger gut zu den Bedürfnissen des Prototypen, denn dieser basiert fast ausschließlich auf CRUD-Operationen. Zudem sind in STANLY_ACOS generell die Geschäftsregeln datenbankseitig implementiert (siehe Abschnitt 3.2). Weiterhin passt dies von der Struktur her überhaupt nicht zu dem URL-Schema des BNL-Moduls (siehe Abschnitt 5.6) und würde entsprechende Übersetzungsmechanismen von dem Parameter *table* zu der entsprechenden Model-Klasse benötigen.

Im Prototyp wird daher eine andere Strategie verfolgt: Es gibt nur eine einzige Model-Klasse, die die CRUD-Operationen generisch bereitstellt und für die die konkrete Tabelle nur ein Parameter ist:[39]

- server/app/
  - Model/
    - Crud.php

Dieses CRUD-Model behandelt die von ExtJS standardmäßig übermittelten Parameter für das Sortieren und Paging (Abschnitt 5.5). Weiterhin werden mehrteilige Primärschlüssel unterstützt, auch wenn im BNL-Modul derzeit nur einteilige Primärschlüssel zum Einsatz kommen (siehe Anhang D). Die benötigten Strukturinformationen über die Tabellen werden direkt von dem `information_schema` der Datenbank abgefragt.

5.15. Integration der Server-Komponenten und Rechteverwaltung

Die neu entstandenen Komponenten des Prototypen müssen in das vorhandene Gesamtsystem von STANLY_ACOS integriert werden. Dies funktioniert aus verschiedenen Gründen nicht vollautomatisch. Serverseitig ist die Integration im Prinzip an zwei Stellen nötig, in der Applikation und in der Datenbank (siehe Abbildung 7 in Abschnitt 3.2).

Für die Applikation ist jedoch keine Integrationsarbeit zu leisten, da die Applikations-Klassen im dort verwendeten Zend-Framework [Zen14] keine explizite Registrierung benötigen, sondern automatisch nach einem strengen Namensschema geladen werden, das sich jeweils aus dem Klassennamen bzw. der Aufruf-URL ableitet (siehe Abschnitt 5.6).

Für die Datenbank hingegen werden mehrere Einträge zur Integration benötigt. Zunächst muss das zentrale Datenbankskript `upgrade.sql` um einen Eintrag für das BNL-Modulskript `bnl/upgrade_module.sql` erweitert werden, welches alle in Abschnitt 5.4 und Abschnitt 5.9 beschriebenen Tabellendefinitionen und Datenbankfunktionen in der richtigen Reihenfolge einlädt:

[39] Dieser wird aus Sicherheitsgründen gegen eine fest definierte Menge von Tabellen geprüft. Konkret im Prototyp sind dies die in Abschnitt 5.4 aufgeführten BNL-Tabellen.

- server/db/
  - upgrade.sql
  - bnl/
    - upgrade_module.sql

Weiterhin wird die in STANLY_ACOS vorhandene Rechteverwaltung für das BNL-Modul erweitert. Dazu wird zunächst eine neue Permission *bnl* eingefügt, deren Vorhandensein im zentralen Eintrittspunkt des serverseitigen BNL-Controllers (Abschnitt 5.6) überprüft wird. Dann wird eine neue Benutzer-Rolle *BNL* definiert, die genau diese Permission sowie die Permission *monitor* beinhaltet. Letztere erlaubt das Betrachten (aber nicht Verändern) der militärische Luftraumbuchungen in STANLY_ACOS. Diese Einträge für die Rechteverwaltung werden in den folgenden Dateien vorgenommen:

- server/db/
  - auth/
    - table_fix/
      - permission.sql
      - role.sql
      - role_permission.sql

Zuletzt wird ein Beispiel-Nutzer *bnl* angelegt, dem die Rolle *BNL* und zudem die Rolle *Single User* zugewiesen wird. Letztere bewirkt, dass der Nutzer nicht für jede Änderung seine Initialien eingeben muss, was in STANLY_ACOS sonst von jedem Nutzer verlangt wird. Der Beispiel-Nutzer wird in der zentralen Beispieldaten-Datei eingetragen:

- server/db/
  - init_demo.sql

5.16. Integration der Client-Komponenten

Auch clientseitig müssen die neu entstandenen Komponenten des Prototypen in das vorhandene Gesamtsystem von STANLY_ACOS integriert werden.

Hierzu wird zuerst der Controller des BNL-Moduls (Abschnitt 5.10) in der folgenden zentralen Applikationsdatei registriert:

- client/App/
  - App.js

Dann wird ein neuer Menüeintrag *BNL* hinzugefügt, über den der Haupt-View des BNL-Moduls (Abschnitt 5.8) aufgerufen werden kann. Dieser Menüeintrag ist so konfiguriert,

dass er nur für Nutzer mit der Permission *bnl* (siehe Abschnitt 5.15) sichtbar ist.[40] Der Eintrag erfolgt direkt in der View-Komponente des Menüs im Haupt-Modul:

- client/App/
  - module/main/
    - view/
      - Menu.js

Es werden also lediglich der Controller und der Haupt-View des BNL-Moduls registriert. Alle weiteren Komponenten des BNL-Moduls werden über das Abhängigkeitssystem des ExtJS-Frameworks geladen.[41]

[40] Dies dient nur der Benutzerfreundlichkeit. Die eigentliche Berechtigungsprüfung findet selbstverständlich serverseitig statt.

[41] Der BNL-Controller hängt von den Stores ab, welche wiederum von ihren jeweiligen Models abhängig sind. Der Haupt-View hängt von seinen Unter-Views ab.

6. Schlussfolgerungen und Ausblick

Die ursprünglichen 46 Anforderungen [Har12] boten einen sehr guten und schnellen Einstieg in das Thema. Zudem war die zweite, eigene Anforderungserhebung sehr sinnvoll: Es konnten Definitionslücken und Unklarheiten in insgesamt 8 Anforderungen beseitigt werden. Die übrigen 38 Anforderungen blieben unverändert. Weiterhin sind 9 Anforderungen neu hinzugekommen, davon 3 funktionale und 6 nichtfunktionale.

Die so aufbereitete Anforderungslage ermöglichte eine klare Absteckung des Bereiches der plausiblen Lösungs-Architekturen. Dieser Bereich konnte im Ausschlussverfahren erfolgreich von 6 auf 4 sinnvolle Hauptkomponenten-Strukturen eingegrenzt werden. Die insgesamt 540 Möglichkeiten, diese Strukturen mit Eigenentwicklungen oder Standardkomponenten zu belegen, konnten wiederum im Ausschlussverfahren auf 78 sinnvolle Möglichkeiten reduziert werden.

Die Anforderungen waren ebenfalls ausreichend, um eine Evaluation aller Architekturen durchzuführen und die am besten geeignete Architektur zu erkennen. Der Prototyp bestätigt voll und ganz die Tragfähigkeit dieser Architektur.

Die erhofften Synergie-Effekte durch Realisierung als Modul in STANLY_ACOS traten überwiegend ein. Die meisten in STANLY_ACOS vorhandenen Mechanismen waren generisch genug, um direkt oder mittels entsprechender Konfiguration für das BNL-Modul von Nutzen zu sein. Jedoch mussten einige Funktionalitäten, wie die automatische Aktualisierung und die Benutzerführung für das BNL-Modul, den spezifischen Anforderungen entsprechend erneut implementiert werden. Dies war jedoch zu erwarten, da es gerade bei solch komplexen Mechanismen keineswegs offensichtlich ist, wie diese sinnvoll verallgemeinert werden sollten, um unterschiedlichen Anforderungen gerecht zu werden.

Zusätzliche Schwierigkeiten bereiteten verschiedenste Probleme in dem von STANLY_ACOS clientseitig verwendeten ExtJS-Framework. Diese reichten von einem Default-Wert, der unnötigerweise browserspezifisch war, bis hin zu einem echten Bug im Framework. Letzterer wurde bereits im April 2013 öffentlich dokumentiert, ist aber in aktuellen ExtJS-Versionen immer noch präsent. Diese Ungereimtheiten waren jeweils harmlos, doch ihre ungewöhnlich starke Häufung machte es (im Vergleich zu klassischen GUI-Frameworks wie Qt, GTK+ oder Swing) äußerst schwierig, eine robuste Benutzerführung zu etablieren, die alle Randfälle abdeckt. Eine gegen Ende der Prototyp-Entwicklung durchgeführte Korrektur der Benutzerführung führte erneut zu zahlreichen unerwarteten Seiteneffekten, was klar aufzeigte, dass der Programmcode des clientseitigen Controllers nicht „robust“ im strengen Sinne ist.[1] Letztendlich konnten jedoch alle im Prototyp entdeckten Probleme mit akzeptablem Aufwand gelöst werden.

[1] In *Structure and Interpretation of Computer Programs* [ASS14, S. 191] wird die folgende Definition für *Robustheit* verwendet: Kleine Änderungen in der Spezifikation sollten möglichst nur genauso kleine Änderungen im Programmcode erforderlich machen.

Der Prototyp konnte bereits potentiellen Nutzern präsentiert werden und ist durchweg positiv angenommen worden. Die Nutzer fanden sich auf Anhieb zurecht, erkannten schnell das Potential dieser voll integrierten Arbeitsweise, stießen aber naturgemäß auch an die Grenzen des Prototypen. Das sekundäre Ziel, dass der Prototyp zu einer präsentierfähigen, interaktiven Diskussions-Grundlage für zukünftige Entwicklungen wird, konnte somit ebenfalls erreicht werden.

Wenn es gelingt, für dieses Konzept innerhalb der DFS genügend Zustimmung zu finden, wird die Abteilung *ATM Daten und Dienste* (OA/L) die eigentliche Software-Entwicklung beauftragen und auch steuern. In diesem Fall könnten sowohl diese Diplomarbeit als auch ihr Vorgängerdokument [Har12] als Basis für ein verbindliches Anforderungsdokument dienen.

Aus wissenschaftlicher Sicht wäre es eine interessante Fragestellung, ob der hier vorgestellte offene, systematische Ansatz auch auf einen höheren Detailgrad der Architektur anwendbar ist. Das heißt, ob eine größere Anzahl von Komponenten mit einer größeren Vielfalt an potentiellen Kommunikationswegen noch beherrschbar wäre und zu sinnvollen Ergebnissen führt. In dieser Arbeit führten vor allem die Anforderungen zur Hardware und zur Netzwerkumgebung zu einer deutlichen Reduzierung der Möglichkeiten bereits in der Frühphase. Dennoch brachte sie 78 Kandidaten hervor, deren systematische Evaluation nur noch computergestützt durchführbar war. Hätte diese Arbeit bei einem höheren Detailgrad angesetzt, wäre möglicherweise schon die Aufstellung der Architektur-Kandidaten nur noch computergestützt möglich gewesen.

Weiterhin ist unklar, ob die hier gewählte Vorgehensweise, bei der die jeweiligen Optionen möglichst frühzeitig reduziert wurden, wirklich notwendig war. Eine alternative Vorgehensweise wäre, den Raum der Möglichkeiten zunächst weit ausufern zu lassen und erst im letzten Schritt, in der Evaluation, sämtliche Ansprüche auf einmal einzubringen. Diese Strategie könnte zu einer noch viel größeren Offenheit gegenüber unerwarteten Lösungsmöglichkeiten führen. Sie birgt aber auch die Gefahr des Verlustes der Übersichtlichkeit. Zudem könnte die Anzahl der Möglichkeiten so weit ausufern, dass sich algorithmische Fragestellungen der effizienten Auswertung ergeben. Der in dieser Arbeit für die Evaluation verwendete Brute-Force-Ansatz wäre dann wahrscheinlich nicht mehr geeignet.

Zuletzt stellt sich die Frage, ob solch ein offener Ansatz tatsächlich zu überraschenden Lösungs-Architekturen führen kann. In dieser Arbeit war die Ergebnis-Architektur zwar nicht von vornherein offensichtlich, aber auch keine große Überraschung. Lediglich die Idee der Peer-to-Peer-Verbindungen (Architektur ES/*) enstand erst während der systematischen Untersuchung, konnte sich jedoch in der Evaluation nicht gegen die naheliegenderen Alternativen durchsetzen. Denkbar wäre eine Anwendung dieses Ansatzes auf möglichst unterschiedliche Software-Projekte, um zu untersuchen, ob dies auch bei anderen Projekten zu interessanten, unerwarteten Lösungen führt, die den naheliegenden Lösungen möglicherweise sogar überlegen sind.

A. Inhalt der DVD

Auf der beigefügten DVD befinden sich folgende Dateien und Verzeichnisse:

bnl.pdf
: ist die Diplomarbeit im PDF-Format mit funktionsfähigen Hyperlinks.

bnl.diff.gz
: ist der Quelltext des Prototypen als gzip-komprimierter Patch. Dieser enthält alle neu angelegten Dateien sowie sämtliche Änderungen an den bestehenden Dateien von STANLY_ACOS in der Version 77843b0 [DFS13b]. Der Patch kann mit dem folgendem Kommando auf das Hauptverzeichnis von STANLY_ACOS angewendet werden:

```
zcat bnl.diff.gz | patch -p1
```

bnl/
: ist der Quelltext des Prototypen in entpackter Form. Dieses Verzeichnis ist inhaltlich identisch mit `bnl.diff.gz` und zur Erleichterung des Reviews gedacht. Alle neu angelegten Dateien sind direkt enthalten und alle Änderungen an bestehenden Dateien befinden sich in einer zusätzlichen Patch-Datei `integration.diff`:

- bnl/
  - client/...
  - server/...
  - integration.diff

bnl.tgz
: ist das Verzeichnis `bnl/` als gzip-komprimiertes Tar-Archiv.

Die Prüfsummen (SHA-256) der Dateien sind:

```
8b961edd846b07af56a51dc6b87f4343dce50a7ee3b76d475fbc3ac08c907d20 bnl.diff.gz
5b4d98b65c2a101fd680828810ed78c6ffe3febc81902c46fe80cc758be5d9d4 bnl.tgz
3243f6a8885a308d313198a2e03707344a4093822299f31d0082efa98ec4e6c8 bnl.pdf
```

B. STANLY_ACOS-Lizenz

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- Neither the name DFS nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.
- **Any modification to the source code MUST be made available to DFS and contributors free of charge and under the same conditions as stated in this license.**

C. System-Umgebung

C.1. System-Voraussetzungen

Der Prototyp stellt folgende Ansprüche an die Hardware oder die virtuelle Umgebung:

CPU-Kerne	CPU	RAM	Festplatte
1	1 GHz	512 MiB	3 GiB

Damit erfüllt er Anforderung 42 problemlos.

Für die vom Prototyp verwendeten STANLY_ACOS-Funktionalitäten werden folgende System-Komponenten benötigt:

Aufgabe	Komponente	Version
Web-Server	Apache/Nginx/…	—
XMPP-Server	Ejabberd	$\geq$ 2.1
Datenbank	PostgreSQL	$\geq$ 9.2[1]
Datenbank	PostGIS	$\geq$ 2.0
Karten-Server	MapServer	$\geq$ 6.0
Prozesskontrolle	Supervisor/Daemontools/…	—
Programmiersprache	PHP	$\geq$ 5.4
Programmiersprache	Python	$\geq$ 2.7

Weitere systemunabhängige Komponenten werden durch das *Clickenv*-Repository bereitgestellt. Darin sind folgende für den Prototypen relevante Pakete enthalten:

Bereich	Aufgabe	Komponente	Version
Server	Framework	Zend-Framework	1.11.4 + 3 Patches
	Karten-Cache	MapProxy	1.1.2
Client	Framework	ExtJS	4.2.1
	XMPP-Client	strophejs	1.0.2
	Karten-Anzeige	OpenLayers	2.12
	Karten-Anzeige	geoext2	bb2add736d
	Karten-Anzeige	proj4js	1.0.2

[1] STANLY_ACOS macht intensiven Gebrauch von der JSON-Unterstützung der PostgreSQL-Datenbank, die erst mit Version 9.2 eingeführt wurde.

C.2. Vorbereitete virtuelle Maschine

Um eine konstante Systemumgebung sicherzustellen, gibt es mit *minstall* eine 1,9 GiB große virtuelle Maschine (komprimiert 420 MiB), die ein Debian/Stable[2] Betriebssystem mit allen benötigten Paketen und Administrations-Werkzeugen enthält. Sie beinhaltet zudem ein angepasstes Paket der PostgreSQL-Datenbank und ihrer Geoinformations-Erweiterung PostGIS, da Debian/Stable derzeit nur mit PostgreSQL 9.1 und PostGIS 1.5, ausgestattet ist, während wie in Abschnitt C.1 beschrieben PostgreSQL 9.2 und PostGIS 2.0 benötigt werden.

Die VM enthält viel mehr, als für den Betrieb des reinen Prototypen benötigt wird. Sie ist jedoch die schnellste Möglichkeit, den Prototypen einzurichten und zu starten:

1. Login per SSH oder Konsole als *root* mit Passwort *test*
2. Neues Passwort für root, etc. setzen:

```
pwsys
```

3. Zugangsdaten für Git-Repository eintragen:

```
pwapps
```

4. Systemunabhängige Bibliotheken installieren:

```
install-app clickenv master
```

5. BNL-Prototyp installieren (Applikation *stanly_acos* im Branch *bnl*):

```
install-app stanly_acos bnl
```

Der Prototyp kann dann über folgende URL im Browser aufgerufen werden:

http://*Name der VM*/stanly_acos_bnl/

Innerhalb der VM liegen die Webapplikationen in folgendem Ordner:

/opt/apps/

Diese liegen dort in separaten Ordnern, deren Namen der Konvention *Applikation_Branch* folgen. Der BNL-Prototyp liegt entsprechend unter:

/opt/apps/stanly_acos_bnl/

Die darunter liegende Ordnerstruktur wurde bereits in Abschnitt 5.3 beschrieben.

[2] Die stabile Version ist derzeit Debian 7.0 „Wheezy“.

D. Unerwartete Probleme

Während der Implementierung des Prototypen in Kapitel 5 tauchten naturgemäß einige Probleme auf. Neben den typischen, selbstverschuldeten Programmierfehlern gab es jedoch auch unerwartete Probleme, die durch Framework- oder System-Komponenten verursacht wurden, die entweder gar nicht funktionierten, oder zumindest nicht wie erwartet.

Die Auswirkungen waren sehr unterschiedlich. Sie reichten von sehr auffälligen Störungen bis hin zu äußerst subtilen Verhaltensfehlern im Prototypen, die nur mit erheblichem Aufwand gefunden und analysiert werden konnten. Eines der Probleme war sogar ein echter Bug im ExtJS-Framework.

Glücklicherweise fand sich für jedes Problem letztendlich eine weitestgehend isolierte, kompakte Lösung in der Größe von etwa 5–20 Zeilen zusätzlichem Programmcode. Keines der Probleme erforderte ein komplettes Redesign von Teilen des Prototypen, oder gar eine Abweichung von der anvisierten Architektur.

Im Folgenden werden die Interessantesten dieser unangenehmen Überraschungen genauer beschrieben.

PostGIS: Deaktivierter GeoJSON-Parser
: Die PostGIS-Erweiterung in der verwendeten virtuellen Maschine (Abschnitt C.2) wurde versehentlich ohne die Bibliothek *JSON-C* compiliert, wodurch datenbankseitig die Funktion *ST_GeomFromGeoJSON* nicht zur Verfügung steht. Somit kann die Datenbank keine Geometrien im GeoJSON-Format parsen, sondern nur als GeoJSON serialisieren. Clientseitig wird daher das ältere WKT-Format anstelle von GeoJSON als interne Repräsentation sowie als Austauschformat verwendet.

ExtJS Bug: Selektiertes Listenelement ist veraltete Model-Instanz
: Fragt man in ExtJS nach dem Neubefüllen eines Stores das selektierte Listenelement eines zugehörigen Grids ab, so erhält man eine veraltete Instanz dessen Models und damit falsche Daten. Dies ist ein Bug im ExtJS-Framework, der bereits April 2013 öffentlich dokumentiert wurde, inklusive konkretem Korrekturvorschlag [Sen13a], aber immer noch in der aktuellen Framework-Version präsent ist. Die öffentlich beschriebene Lösung diente als Konzept für einen ähnlichen Workaround im Prototypen.

ExtJS: Mehrteilige Primärschlüssel
: Das ExtJS-Framework bietet keine Unterstützung für mehrteilige Primärschlüssel. Dies wäre zum Beispiel für die BNL-Vorhaben nützlich, deren natürlicher Schlüssel aus dem Jahr und der laufenden Nummer besteht. Als Workaround wurde eine künstliche ID-Spalte eingeführt. Mit Blick auf den externen Server und Client fiel die Entscheidung auf die Verwendung von UUIDs statt einer Sequenz, da UUIDs auch außerhalb der Datenbank generiert werden können und stets eindeutig sind.

ExtJS: Auslesen von Formulardaten und Validierung

Die Methode *getValues()* der Klasse *Ext.form.Panel*, die zum Auslesen der eingegebenen Daten eines Formular dient, liefert bei Nicht-Textfeldern unerwartete Werte. Bei Checkboxen etwa liefert sie „on“ / *undefined* anstelle von *true* / *false*. Hier wird offenbar das Verhalten von reinen HTML-Formularen nachempfunden. Dieses unerwünschte Verhalten kann auf zwei Weisen abgeschaltet werden: einerseits über den vierten optionalen Parameter (*useDataValues*) von *getValues()* und andererseits über die Methode *getFieldValues()*, die jedoch nicht direkt in der Komponente *Ext.form.Panel* verfügbar ist, sondern nur in der deren Teilkomponente *Ext.form.Basic*.

ExtJS: Datumsformat beim Laden von JSON-Daten

Die Konfigurationsoption *dateFormat* der Klasse *Ext.data.Field* ist optional, das Standardverhalten ist dann jedoch browserspezifisch. Daher sollte diese Option im Zweifel stets von Hand auf den Wert „c“ (ISO-Format) gesetzt werden.

ExtJS: Dynamische Default-Werte

In der Klasse App.module.bnl.model.Bnl soll es dynamische Default-Werte geben, zum Beispiel, dass der voreingestellte Zeitraum der jeweils aktuelle Tag ist, auch wenn die Applikation mehrere Tage lang am Stück gelaufen ist. Dies wird vom ExtJS-Framework jedoch nicht direkt unterstützt. da es keine definierte Schnittstelle für dynamische Default-Werte gibt. Im Prototyp wird dieses Problem dadurch gelöst, dass dynamische Objekt-Properties (*Object.defineProperty*) verwendet werden. Dieses Sprach-Feature von JavaScript wurde mit ECMAScript 5 eingeführt.

E. Automatisch generierte Tabellen

Einige Tabellen dieser Arbeit sind automatisch generiert:

1. Zusammenfassung der Kombinationen von Hauptkomponenten (Abschnitt 3.4)
2. Auswertungstabelle (Abschnitt 4.3)
3. Kurze Auswertungstabelle der Maxima (Abschnitt 4.3)

Dies wird durch das nachstehende Python-Programm bewerkstelligt. Als Eingabe dienen verschiedene Tabellen dieser Arbeit, die direkt aus dem LaTeX-Quelltext extrahiert werden.

```
import compiler.ast
import re
import StringIO

def parse_summary(tex):
    table_regex = r'Arch\. *&.*?\n(.*?)\n\\end{tabular}'
    line_regex = r'^ *(.*?)/\* *& *(.*?) *& *(.*?) *& *(.*?) *& *(.*?) *& *(.*)$'
    return [[(arch, _is.split(','), es.split(','), ic.split(','), ec.split(','), rest)
             for arch, _is, es, ic, ec, rest in re.findall(line_regex, table, re.M)]
            for table in re.findall(table_regex, tex, re.S)]

def group_by_first(lines):
    cur = None
    result = []
    for i, line in enumerate(lines):
        if i != 0 and line[0] != cur:
            yield cur, result
            result = []
        cur = line[0]
        result.append(line[1:])
    yield cur, result

def parse_criteria(tex):
    section = r'\section{Kriterien}'
    table_regex = r'Krit\. *&.*?\n(.*?)\n\\end{tabular}'
    line_regex = (r'^ *(.*?) *& *(.*?) *& *(.*?) *& *(.*?) *& *(.*?) *& *(.*?) *'
                  + r'& *(.*?) *\\\\$')
    return list(group_by_first([
        (crit_name, [pat_ar, pat_is, pat_es, pat_ic, pat_ec], int(value))
        for table in re.findall(table_regex, tex.split(section)[1], re.S)
        for crit_name, pat_ar, pat_is, pat_es, pat_ic, pat_ec, value
            in re.findall(line_regex, table, re.M)
    ]))

def crit_max(crit):
    return max(value for patterns, value in crit)

def criteria_max(criteria):
    return [(name, crit, crit_max(crit)) for name, crit in criteria]
```

```
def pattern_match(pattern, field):
    inner_regex = pattern.replace(',', '|').replace('*', '.*').replace('\\', r'\\')
    return re.match('^(' + inner_regex + ')$', field) is not None

def crit_match(crit_name, crit, fields):
    values = [value for patterns, value in crit
                    if all(pattern_match(p, f) for p, f in zip(patterns, fields))]
    if len(values) == 1:
        return values[0]
    raise Exception('%d matches for %r on %r' % (len(values), crit_name, fields))

def build_archlist(summary, criteria):
    return [(fields,
             [(name, crit_match(name, crit, fields), m) for name, crit, m in criteria])
            for fields in ((arch, val_is, val_es, val_ic, val_ec)
                           for table in summary
                           for arch, list_is, list_es, list_ic, list_ec, rest in table
                           for val_is in list_is
                           for val_es in list_es
                           for val_ic in list_ic
                           for val_ec in list_ec)]

def better(arcrit1, arcrit2):
    return (all(val1 >= val2 for (_, val1, _), (_, val2, _) in zip(arcrit1, arcrit2))
            and arcrit1 != arcrit2)

def arch_localmax(arcrit, archlist):
    return not any(better(arcrit_other, arcrit) for _, arcrit_other in archlist)

def archlist_localmax(archlist):
    return [(ar, arcrit, arch_localmax(arcrit, archlist)) for ar, arcrit in archlist]

def format_summary(summary, archlist):
    s = StringIO.StringIO()
    print >>s, r'\begin{tabular}{lccccrr}'
    print >>s, r'Arch. & Int.~Server & Ext.~Server & Int.~Client & Ext.~Client',
    print >>s, r'& \# & $\Sigma$ \\'
    print >>s, r'\hline'
    for table in summary:
        for arch, list_is, list_es, list_ic, list_ec, rest in table:
            print >>s, '{}/*'.format(arch),
            for list_x in [list_is, list_es, list_ic, list_ec]:
                print >>s, '&', ','.join(list_x),
            print >>s, '&', rest
        print >>s, r'\hline'
    print >>s, r'& & & & & & {} \\'.format(len(archlist))
    print >>s, r'\end{tabular}'
    return s.getvalue()

def format_archlist(archlist, criteria, short):
    s = StringIO.StringIO()
    env = 'tabular' if short else 'longtable'
    print >>s, r'\begin{' + env + '}{l' + ('c' * len(criteria)) + 'c}'
    print >>s, r'Architektur'
    for crit_name, crit, crit_max in criteria:
        print >>s, '&', crit_name
    print >>s, r'& Max'
    print >>s, r'\\ \hline' if short else r'\\ \hline \endhead'
    for fields, arcrit, localmax in archlist:
        if localmax or (not short):
            print >>s, '{:17s}'.format('{}/{}-{}-{}-{}'.format(*fields)),
```

```
            for crit_name, val, crit_max in arcrit:
                if short:
                    print >>s, '&', '{:3d}'.format(val) if val != crit_max else r'---',
                else:
                    print >>s, '&', val,
            print >>s, r'& \yes \\' if localmax else r'& --- \\'
    print >>s, r'\end{' + env + '}'
    return s.getvalue()

with open('architekturen.tex') as f:
    summary = parse_summary(f.read())
with open('evaluation.tex') as f:
    criteria = criteria_max(parse_criteria(f.read()))
archlist = archlist_localmax(build_archlist(summary, criteria))
with open('autogen_summary.tex', 'w') as f:
    f.write(format_summary(summary, archlist))
with open('autogen_archlist.tex', 'w') as f:
    f.write(format_archlist(archlist, criteria, short=False))
with open('autogen_archlist_short.tex', 'w') as f:
    f.write(format_archlist(archlist, criteria, short=True))
```

Abbildungsverzeichnis

Literaturverzeichnis

[Amt13] Amt für Flugsicherung der Bundeswehr: *Luftraummanagement COMIL & AMC*. http://www.afsbw.de/index.php/flusi/luftraum-a-flugbetrieb. Version: Februar 2013, Abruf: 09.02.2013, SHA-256: 3434a141cd4ab9f2ea89e3502bdb725483561de3a402e13d37214aefccb08c90

[ASS14] Abelson, Harold ; Sussman, Gerald J. ; Sussman, Julie: *Structure and Interpretation of Computer Programs*. https://github.com/sarabander/sicp-pdf/raw/59c66cb245a5fd91b81a4286844a94da60016e6b/sicp.pdf. Version: Unofficial Texinfo Format 2.andresraba5.3, April 2014, Abruf: 12.05.2014, SHA-256: 978daaab607fbbff9a7dbffc136605c7cd2c2325143cd2cdea5a1644dd10ed52

[Ber07] Bernstein, Daniel J.: *Some Thoughts on Security After Ten Years of Qmail 1.0*. http://cr.yp.to/qmail/qmailsec-20071101.pdf. Version: November 2007, Abruf: 21.03.2014, SHA-256: 5db0b9a8f746c8cbab1ab967b686fa33635935460d1aab6e8e29c4626559b59a

[Bro75] Brooks, Frederick P.: *The Mythical Man-Month: Essays on Software Engineering*. Addison-Wesley, 1975 https://archive.org/download/mythicalmanmonth00fred/mythicalmanmonth00fred.pdf. – ISBN 0–201–00650–2. – SHA-256: 9a04cc52791b1b7d9631d2f7a83b32adff72c661cfe857efa1ea83bd15f57d41

[Bro95] Brooks, Frederick P.: *The Mythical Man-Month: Essays on Software Engineering, Anniversary Edition*. Addison-Wesley, 1995 http://books.cat-v.org/computer-science/mythical-man-month/tmmm.pdf. – ISBN 0–201–83595–9. – SHA-256: 04d8a1a2c2227a8a3edfda04e91d1058b94957b35ee71867af39e8386d46c879

[Bun12] Bundesministerium für Verkehr: *Luftverkehrs-Ordnung (LuftVO) in der Fassung der Bekanntmachung vom 27. März 1999 (BGBl. I S. 580), die durch Artikel 3 des Gesetzes vom 8. Mai 2012 (BGBl. I S. 1032) geändert worden ist*. http://www.gesetze-im-internet.de/bundesrecht/luftvo/gesamt.pdf. Version: Mai 2012, Abruf: 01.08.2012, SHA-256: 399850a00efd687dac6b552164a06b02de36a7c82f912272f7854d7db4ae9f53

[Bur06] Bureau International des Poids et Mesures (BIPM): *The International System of Units (SI)*. http://www.bipm.org/utils/common/pdf/si_brochure_8_en.pdf. Version: 8, Mai 2006, Abruf: 17.11.2012, SHA-256: 1a9eb8cd04ca4b7f82ac95ff4cbab7119d84bc2222e34b71e3f7575e63ccdf9f

[Cac10] CacheWiki: *OVL*. http://www.cachewiki.de/w/index.php?title=OVL&oldid=10688. Version: Juni 2010, Abruf: 16.11.2012, SHA-256:

c4e0bd200cc1d984bf4b3edf18b51f612e1ed7fe8324626c57ad2389f989e3ae

[Cen06] Central Intelligence Agency (CIA): *The World Factbook – Appendix G.* https://www.cia.gov/library/publications/the-world-factbook/appendix/print_appendix-g.html. Version: 8, Mai 2006, Abruf: 17.11.2012, SHA-256: 8d81baec055881ff30b245af7b1c72d619aa66a58b62bf006d0003b2381cdaee

[Deu12] Deutscher Bundestag: *Luftverkehrsgesetz (LuftVG) vom 1. August 1922 (RGBl. 1922 I S. 681), das zuletzt durch Artikel 1 des Gesetzes vom 8. Mai 2012 (BGBl. I S. 1032) geändert worden ist.* http://www.gesetze-im-internet.de/bundesrecht/luftvg/gesamt.pdf. Version: Mai 2012, Abruf: 01.08.2012, SHA-256: 932ea5e90d8de762128aaffacddc79c9a5cadbd0e36416a13f22938162ab12d0

[DFS07] DFS Deutsche Flugsicherung GmbH: *NfL I 59 / 07 – Voraussetzungen für die Erteilung einer Flugverkehrskontrollfreigabe zur Durchführung von Fallschirmabsprüngen und zum Abwerfen von Gegenständen an Fallschirmen im kontrollierten Luftraum.* Nachrichten für Luftfahrer. 10592007.pdf. Version: März 2007, Abruf: 01.03.2007, SHA-256: 87837f46aa091c7ac4e53130548d6f39e5c81d163df88576ab7b691b173419da

[DFS08a] DFS Deutsche Flugsicherung GmbH: *Military Variable Profile Area in the Bremen FIR / Rhein UIR in Germany.* DMEAN Best Practices Register. https://www.eurocontrol.int/sites/default/files/content/documents/nm/best-practices/bp-dfs-mvpa.pdf. Version: Mai 2008, Abruf: 02.07.2012, SHA-256: 3ca8163ab9e298509dac5b9ea9a9f648436fafca575753734d1d10587d81ca83

[DFS08b] DFS Deutsche Flugsicherung GmbH: *STANLY_MVPA.* DMEAN Best Practices Register. https://www.eurocontrol.int/sites/default/files/content/documents/nm/best-practices/bp-dfs-stanly-tool.pdf. Version: Mai 2008, Abruf: 02.07.2012, SHA-256: 3317645ee91874a49575e45c033fd2bf044c57188ba56ecab8de2d5fcc9b5427

[DFS09] DFS Deutsche Flugsicherung GmbH: *NfL I 32 / 09 – Bekanntmachung über die Festlegung von Mindestvorlaufzeiten für die „Besondere Nutzung Luftraum“.* Nachrichten für Luftfahrer. 10322009.pdf. Version: Februar 2009, Abruf: 12.02.2009, SHA-256: 633e6827032526c70f9f417a50936e82eab72ec408f59cb64071681280723e0d

[DFS11] DFS Deutsche Flugsicherung GmbH: *NfL I 110 / 11 – Bekanntmachung der besonderen Voraussetzungen für die Erteilung von Flugverkehrskontrollfreigaben.* Nachrichten für Luftfahrer. 11102011_sig.pdf. Version: Juli 2011, Abruf: 14.07.2011, SHA-256: b9cfc78ea99d2952e2bcb1986314ec01d6bde82a545fb12951b1b53bb15766d8

[DFS12] DFS Deutsche Flugsicherung GmbH: *Luftraum Berlin – gültig ab 3.6.2012, ICAO 1:500 000.* http://www.dfs.de/dfs/internet_2008/module/presse/deutsch/presse/presseinformation/2012/

neue_luftraumstruktur_berlin_ab_3_juni_in_kraft_20_4/berlin_neuer_luftraum.pdf. Version: April 2012, Abruf: 24.07.2013, SHA-256: cdd26da48fac5b2b0681d97bddf882ddc21707d7a8fcccfd43299a242060b25d

[DFS13a] DFS Deutsche Flugsicherung GmbH: *Geschäftsbericht 2012*. http://www.dfs.de/dfs_homepage/de/Unternehmen/Zahlen%20und%20Daten/Finanzen/gb2012_de.pdf. Version: Juni 2013, Abruf: 28.11.2013, SHA-256: 1b63b9e03aed702966d0724ac76a30950e244602ee5da2526fe73f6b9f7e1f30

[DFS13b] DFS Deutsche Flugsicherung GmbH: *STANLY_ACOS*. stanly_acos_77843b0.tgz. Version: 77843b0 vom 13. Dezember 2013, Dezember 2013, Abruf: 13.12.2013, SHA-256: b54a2ff8b6759b3d830a578c03633497c400450b170195500020d41a37d25e06

[Eur04] Europäisches Parlament und Rat der Europäischen Union: *Verordnung (EG) Nr. 552/2004 des Europäischen Parlaments und des Rates über die Interoperabilität des europäischen Flugverkehrsmanagementnetzes („Interoperabilitäts-Verordnung")*. Amtsblatt der Europäischen Union 96/2004. http://eur-lex.europa.eu/LexUriServ/LexUriServ.do?uri=OJ:L:2005:342:0020:0025:DE:PDF. Version: März 2004, Abruf: 09.07.2013, SHA-256: 2702ef441c3cbed942d36739fdbd0dfe2a6d6ede2f305baf545d7ccdd188add7

[Eur05] Europäische Kommission: *Verordnung (EG) Nr. 2150/2005 der Kommission über gemeinsame Regeln für die flexible Luftraumnutzung*. Amtsblatt der Europäischen Union 342/2005. http://eur-lex.europa.eu/LexUriServ/LexUriServ.do?uri=OJ:L:2005:342:0020:0025:DE:PDF. Version: Dezember 2005, Abruf: 25.06.2013, SHA-256: 18490a3e46f188d9a5bad23433b0329e12a55ab1aad67a86c726c6b947d98477

[EUR09] EUROCONTROL: *„Have a good flight!" – Flexible use of airspace*. Video. http://youtube.com/v/eLqBCwSPT50. Version: November 2009, Abruf: 24.06.2013, WebM, SHA-256: 6426725acce0be9e16a5d287a1ca7f2a1d71405ef16b35ff04be4f80bd888d71

[EUR12a] EUROCONTROL: *Airspace Management Handbook – Guidelines for Airspace Management*. European Route Network Improvement Plan, Part 3. https://www.eurocontrol.int/sites/default/files/content/documents/nm/airspace/ernip-part-3-asm-handbook.pdf. Version: Oktober 2012, Abruf: 10.07.2013, SHA-256: fc8c7d7dc82d83195c200e2859f91a21f7ddeb6a6525f1b37574d769d0b46542

[EUR12b] EUROCONTROL: *FABs – Functional Airspace Block*. https://www.eurocontrol.int/dossiers/fabs. Version: November 2012, Abruf: 16.11.2013, SHA-256: 8d2408124ce52d1a720182b74dda83c022c8de12800587210b17cd5c9900549f

[EUR12c] EUROCONTROL: *Flexible use of airspace*. https://www.eurocontrol.int/articles/flexible-use-airspace. Version: Oktober 2012, Abruf: 24.06.2013, SHA-256: bc49e96c8250b810172509170a8f52809130c1f6bdc8bcbdc88003afd039a697

[Eur12d] Europäische Kommission: *Single European Sky*. http://ec.europa.eu/transport/modes/air/single_european_sky/. Version: Oktober 2012, Abruf: 12.07.2013, SHA-256: 34109b36ace0003925fd2718ca1ce52488eeac5f2fe0b479fb16c2c518e59024

[Fow03] Fowler, Martin: Who Needs an Architect? In: *Software, IEEE* 20 (2003), September, Nr. 5, 11–13. http://martinfowler.com/ieeeSoftware/whoNeedsArchitect.pdf, Abruf: 07.11.2013, SHA-256: 19767622f4245b9c04f65d2a677a96a6c256cbba25f5fc4d66679a026dff5a29

[Har12] Hartwig, Nils: *STANLY_ACOS – BNL Funktionalität*. BNL Anforderungsdokument.pdf. Version: Januar 2012, Abruf: 20.06.2012, SHA-256: cb8d5dea7e529118847f16a7e508add5f3518502ad74355b6fca3c66be690071

[Hor13] Hortmann, Ralf: *STANLY_ACOS – Statistics and Analysis – Airspace Coordination System*. ACOS Vorstellung deutsch Dez 2011.ppt. Version: Juli 2013, Abruf: 25.07.2013, SHA-256: 1011b3f395e71b3dfea26bbc180458190111e633fbfb871f54d6c6a969ec9933

[Ipp05] Ippolito, Bob: *Remote JSON - JSONP*. http://bob.ippoli.to/archives/2005/12/05/remote-json-jsonp/. Version: Dezember 2005, Abruf: 08.06.2014, SHA-256: 8732fce1b37c4fdce198227cb0d987b2e5ece401dda4ab6a0599cfec4beeab77

[Lan12] Landesamt für Geoinformation und Landentwicklung Niedersachsen (LGLN): *Formelsammlung unter Berücksichtigung des amtlichen Bezugssystems ETRS89 mit UTM-Abbildung*. http://www.lgn.niedersachsen.de/download/29844/Formelsammlung_unter_Beruecksichtigung_des_amtlichen_Bezugssystems_ETRS89_mit_UTM-Abbildung.pdf. Version: Juli 2012, Abruf: 23.04.2014, SHA-256: 9baeaae848fccb41f65e9f8398f7d97a0f1b58173f780eb14366ea24a86b2109

[OGP14a] OGP Geomatics Committee: *EPSG:25832*. EPSG Geodetic Parameter Registry. https://www.epsg-registry.org/report.htm?type=selection&entity=urn:ogc:def:crs:EPSG::25832&reportDetail=long&style=urn:uuid:report-style:default-with-code&style_name=OGP%20Default%20With%20Code&title=EPSG:25832. Version: April 2014, Abruf: 23.04.2014, SHA-256: 3f47bd78ceb2a945c7220d216bd8d1e09b15dda1b702b1f188a5705a030fd89a

[OGP14b] OGP Geomatics Committee: *EPSG:3857*. EPSG Geodetic Parameter Registry. https://www.epsg-registry.org/report.htm?type=selection&entity=urn:ogc:def:crs:EPSG::3857&reportDetail=long&style=urn:uuid:report-style:default-with-code&style_name=OGP%20Default%20With%20Code&title=EPSG:3857. Version: April 2014, Abruf: 23.04.2014, SHA-256: 85ad12dfd8f31e61f87b2ed49dcd2d19a94c78349f83a16524037ef7fcb30e10

[OGP14c] OGP Geomatics Committee: *EPSG:4326*. EPSG Geodetic Parameter Registry. https://www.epsg-registry.org/report.htm?type=selection&entity=urn:ogc:def:crs:EPSG::4326&reportDetail=long&style=urn:

uuid:report-style:default-with-code&style_name=OGP%20Default%20With%20Code&title=EPSG:4326. Version: April 2014, Abruf: 23.04.2014, SHA-256: 43c5e0bde213f56bbcc49b203b2e2385e82754161db1ee66a6a40a3e5a8d30cd

[Ope10a] Open Geospatial Consortium (OGC): *OpenGIS Implementation Standard for Geographic information – Simple feature access – Part 2: SQL option*. http://portal.opengeospatial.org/files/?artifact_id=25354. Version: OGC 06-104r4, August 2010, Abruf: 22.04.2014, SHA-256: 6f449d6873b404555aa687010101aa549066087b47125722bfc5ef71026a7bf8

[Ope10b] Open Geospatial Consortium (OGC): *OpenGIS Web Feature Service 2.0 Interface Standard (also ISO 19142)*. http://portal.opengeospatial.org/files/?artifact_id=39967. Version: OGC 09-025r1, November 2010, Abruf: 26.11.2013, SHA-256: a47d34cb1da83fb6e6620fdade6235e623ee2bd6a83af3e5be3c0581a7d71bbf

[Ope10c] OpenStreetMap Wiki: *Nominatim*. https://wiki.openstreetmap.org/w/index.php?title=DE:Nominatim&oldid=491102. Version: Juni 2010, Abruf: 18.11.2012, SHA-256: 8eaa9823fab1dcffa4414c94a3c090ad262fbfc2d674dcae45b20f05eab3ccf8

[Ope12] Open Source Geospatial Foundation (OSGeo): *OpenLayers Documentation*. http://docs.openlayers.org/. Version: Oktober 2012, Abruf: 18.04.2014, SHA-256: bf71c2919cdf41d42aeb45d08384146e8b2eaefb112cef4b9ce655a49f7f5ee2

[Ope14] Open Source Geospatial Foundation (OSGeo): *PostGIS 2.0 Manual*. http://www.postgis.net/stuff/postgis-2.0.pdf. Version: SVN Revision (12480), März 2014, Abruf: 22.04.2014, SHA-256: cbbad614dac9cd6a1adbfa092f611c743e36c6d719b9b46a605adfac87dce5d8

[PSA10] Paterson, Ian ; Saint-Andre, Peter: *XEP-0206: XMPP Over BOSH*. http://xmpp.org/extensions/xep-0206.html. Version: 1.3, Juli 2010, Abruf: 16.11.2013, SHA-256: cae1fc185748f42b2c101555975dcdbe0f3e8c603b2c25637de6f0a118d0d8e4

[PSSAM10] Paterson, Ian ; Smith, Dave ; Saint-Andre, Peter ; Moffitt, Jack: *XEP-0124: Bidirectional-streams Over Synchronous HTTP (BOSH)*. http://xmpp.org/extensions/xep-0124.html. Version: 1.10, Juli 2010, Abruf: 16.11.2013, SHA-256: 77bc2123609dec323072303945e2ea1ae25f559f1839a9bd002e6fe2b99e4abe

[Ray03] Raymond, Eric S.: *The Art of Unix Programming*. http://www.catb.org/esr/writings/taoup/html/graphics/taoup.pdf. Version: September 2003, Abruf: 04.06.2014, SHA-256: be9174b0437bad9e0900d39a76a81bc54dc9e02c42a29e37ea90d85fe7843080

[Ray12] Raymor, Brian: *WebSockets in Windows Consumer Preview*. http://blogs.msdn.com/b/ie/archive/2012/03/19/websockets-in-windows-consumer-preview.aspx. Version: März 2012, Abruf: 07.04.2014, SHA-256: bece802ae5ceaf82da99d928ea6dcf80ae5f0508c03b49b01e6c28ae793b30dd

[Sen13a] Sencha: *ExtJS: Outdated selection after grid store reload (Open Bug without ID).* http://www.sencha.com/forum/showthread.php?261027. Version: April 2013, Abruf: 24.03.2014, SHA-256: bfea74f4d7d2a1c72c965ad3f44a99a20dffa827ceac08dd545d287b87d7769f

[Sen13b] Sencha: *ExtJS: Performance issues when loading associations (Open Bug EXTJSIV-9793).* http://www.sencha.com/forum/showthread.php?263285. Version: September 2013, Abruf: 24.03.2014, SHA-256: 27358fcf0e16887a738f43b91e8082cfdcd099a359491ba22986ecdc72ff0270

[The14] The PostgreSQL Global Development Group: *PostgreSQL 9.2.8 Documentation (A4).* https://www.postgresql.org/files/documentation/pdf/9.2/postgresql-9.2-A4.pdf. Version: März 2014, Abruf: 27.03.2014, SHA-256: 8ccee21d253155f3b091230598efa8a782227debc18afddbe80acdb6b5eadb26

[Wes13] Westdeutscher Rundfunk Köln (WDR): *Die WDR Computer-Nacht.* Video. https://www.youtube.com/watch?v=aQAeJAEgG4Q. Version: April 2013, Abruf: 01.04.2014, MP4, fmt22_720p, SHA-256: 8faae2db798738d7aac1d7fefd8da96d693cc3000e3da95c395adf0090e45030

[Wor12] World Wide Web Consortium (W3C): *The WebSocket API.* http://www.w3.org/TR/2012/CR-websockets-20120920/. Version: Candidate Recommendation 2012-09-20, September 2012, Abruf: 07.04.2014, SHA-256: db4f26491b2bc0d23eb36d1e09488e5e3dbbc3d5d13317cbb52bcfd92953c207

[Wor14] World Wide Web Consortium (W3C): *Cross-Origin Resource Sharing.* http://www.w3.org/TR/cors/. Version: Recommendation 2014-01-16, Januar 2014, Abruf: 08.06.2014, SHA-256: e3f7697e1a9d53d33065512fc04dafe10f118356841fcdeaa6260a613f9ea523

[Zen14] Zend Framework Community: *Zend Framework 1.11 Reference Guide.* http://framework.zend.com/manual/1.11/en/manual.html. Version: März 2014, Abruf: 03.03.2014, SHA-256: e121df3f8f8a92f8641e351deda1de4d6bbfcb7d8d89757efd75aa7607ff3125